Improved Robust Secret-Sharing for Honest-Majority Preprocessing-MPC

Charanjit S. Jutla[1], Nathan Manohar[1], and Arnab Roy

IBM T.J. Watson Research Center

Abstract. We design a novel robust secret-sharing scheme in the honest-majority setting for n parties that allows constant communication per opening per party when amortized over $\Theta(n)$ simultaneous openings. We attain this scheme by a novel application of list decoding of folded Reed-Solomon codes. This was stated as an open problem by Escudero and Fehr (2023), who could obtain this amortized complexity only for $\Theta(n^2)$ simultaneous openings. Additionally, the dealing of such shares for a random value in a finite field $\mathbb{F}$ can itself be done using an honest-majority MPC protocol using only $O(n^2)$ multiplications between Shamir-shared values.

This allows us to build an offline-online fully-secure (GOD) honest-majority MPC protocol, which has a fast online stage with amortized communication cost only $O(n)$ per multiplication gate (assuming the circuit has width $\Omega(n)$) and with no additional overhead, and an offline preprocessing stage with incremental communication cost only $O(n^3)$ per multiplication gate. This improves the previous best protocol in this setting by Escudero and Fehr (2023), which required the circuit to have width $\Omega(n^2)$. Further, the earlier scheme had an incremental preprocessing cost of $O(n^4)$ per multiplication gate.

Due to requiring non-responsive rounds, the online stage of these protocols may still be concretely far less efficient than the online stage of the offline-online BGW protocol. Since the BGW protocol only provides security against fewer than $n/3$ malicious parties, we also devise a novel MPC protocol that boosts the BGW protocol's correctness threshold to $n/2$, while maintaining privacy and robustness thresholds at $n/3$, and with minimal online overhead. This is useful in applications, such as elections and auctions, where certification of a correct result is paramount. To achieve this correctness against fewer than $n/2$ malicious parties and to keep the pre-processing cost low, we use our novel robust honest-majority secret-sharing scheme, but only once during a final SPDZ-style attestation.

Keywords: MPC · Robust Secret-Sharing · Offline-Online · List Decoding

Table of Contents

1 Introduction 4
1.1 Our Main Contribution 5
1.2 An Additional Contribution 6
1.3 On the $t < n/3$ vs. $t < n/2$ Settings 7
1.4 Related Work 8
2 Technical Overview 9
2.1 Our Novel Robust Secret-Sharing Scheme 9
2.2 Efficient $\Theta(n)$ Simultaneous Openings via List Decoding 11
2.3 Enhancing the Correctness of Offline-Online BGW 12
3 Preliminaries 13
3.1 Secret Sharing Preliminaries 14
3.2 Opening Sharings for $t < n/3$ 15
3.3 Random Affine Functions 15
4 A Novel Robust Secret-Sharing Scheme 16
4.1 Privacy: View of The Minority Adversary in Any Encoding. 17
4.2 Simple Robust (Single) Opening 18
4.3 Efficient Simultaneous Openings a la [26] and [29] 19
5 $\Theta(n)$ Simultaneous Robust Openings Using Three-Folded List Decoding 20
5.1 The Three-Folded Code and List-Decoding Bound 20
5.2 Protocol for Efficient Simultaneous Robust Opening of $t + 1$ Sharings 21
6 Ideal Functionality for Extra-Correct Secure-Function-Evaluation (X-SFE) 23
7 Input Sharing 23
8 Our Honest-Majority Fully-Secure MPC Protocol 24
9 A Novel $n/3$-Private and Robust MPC Protocol with Honest-Majority Correctness 24
9.1 UC Security Proof 27
10 Preprocessing Phase 27
10.1 Introduction to "Guaranteed Output Delivery Comes Free in Honest Majority MPC" [39] 27
10.2 Honest-Majority Protocols 28
A Proof of Privacy [Theorem 4.2] 33
B Protocol and Proof of Robust (Simultaneous) Opening [Theorem 4.5]. 35
C Deferred Proofs of Section 5 39
C.1 Proof of Lemma 5.2 39
C.2 Proof of Lemma 5.3 40
C.3 Proof of Thm. 5.4 41
D Ideal Functionality for Extra-Correct SFE 41
E Input Sharing 42

E.1 Ideal Functionality for Robust Secret-Sharing 42
E.2 Ideal Functionality for Input Sharing 43
E.3 UC Realization of $\mathcal{F}_{\mathsf{inputShare}}$ in the Preprocessing Model 45
F Ideal Functionality $\mathcal{F}_{\mathsf{Beaver}}$ 48
G Ideal Functionality X-Beaver 49
H Proof of Theorem 9.1 ($\mathsf{ONLINE-X\text{-}SFE}$) 50
I Extension of Thm. 1.3 with Honest-Majority Robustness 56
J Preprocessing Analysis for Honest-Majority Protocol (Thm. 1.2) 56
K Preprocessing Analysis for Enhanced-$n/3$ MPC Protocol 57
K.1 Enhanced-Correctness $n/3$ Protocol 57
K.2 Complexity Analysis of $\mathcal{F}_{\mathsf{X\text{-}Beaver}}$ Instantiation 60
L Analysis of Preprocessing Complexity of Robust Shares of [29] 60
M Concrete Example 62

1 Introduction

Secure multi-party computation (MPC) is a fundamental cryptographic primitive that allows a set of mutually distrusting parties $P_1, \ldots, P_n$, each possessing an input x_i, to compute a function $f(x_1, \cdots, x_n)$ on their joint inputs without revealing anything other than the output of the function. Such protocols are said to be *maliciously secure* if this holds even against an adversary that corrupts t of the parties and has them deviate arbitrarily from the protocol. In this work, we focus on maliciously secure protocols in the honest-majority setting, where $t < n/2$. The honest-majority setting is of particular interest because in this setting, it is possible to achieve *guaranteed output delivery* (GOD), which means that the honest parties receive the output of the computation regardless of the behavior of the t corrupted parties. This is particularly important in settings where it is crucial that the protocol does not fail. In the honest-majority setting, security can be achieved against a computationally unbounded adversary (perfect security in the case $t < n/3$ [12, 11] and statistical security in the case $t < n/2$ [50, 7, 23, 10, 13, 39], assuming a necessary broadcast channel).

Since the early days of advances in secure multi-party computation (MPC), much research has focused on protocols that can be split into an *offline stage*, also known as a preprocessing stage, and a highly efficient *online stage* [8]. The offline stage generates correlated randomness for the parties before the parties have the inputs to the protocol. When the inputs are set, the parties can run a relatively fast online stage, capitalizing on the correlated randomness. We will refer to such protocols as *preprocessing MPC*. Essential to many preprocessing MPC protocols is the idea of precomputed shares of Beaver multiplication-tuples [8]. The "sharing" determines the correlated randomness, and, typically, the sharing employs a linear code, e.g. Shamir sharing [51] (Reed-Solomon encoding) or even just a fully additive-sharing [27]. In this work, we focus on information-theoretically secure MPC protocols with GOD in the preprocessing model with an online stage requiring $O(n|C|)$ total communication and $O(D)$ rounds, where $|C|$ is the size of the circuit to be computed measured in the number of multiplication gates and D is the depth of the circuit. Such protocols have communication per party and rounds independent of n, which is desirable for scalability.

The first two-thirds honest-majority MPC protocol [12] made a remarkable use of Shamir sharing to show that not only is the protocol private but it also achieves robustness, i.e. all honest parties are guaranteed correct output delivery (for instance using a Reed-Solomon decoder). For two-thirds honest-majority preprocessing MPC, the Shamir sharing inherent in the precomputed Beaver-tuples also allows a fast "opening" of shares[1] required per gate in the online stage. In fact [26] showed a two round opening protocol that only requires communicating a small constant number of field elements per opening per party, *if amortized over* $\Theta(n)$ *simultaneous openings* (say, for all gates at the same depth

[1] These shares are typically of output wire values of a gate, but blinded (inductively) by a further random value.

of a circuit). Thus, in the $t < n/3$ setting, it is known how to achieve an online stage with $O(n|C|)$ total communication and $O(D)$ rounds [12, 11].

Adapting this approach to the honest-majority setting faces some immediate challenges. For honest-majority preprocessing MPC, the Shamir-sharing must now use shares that are evaluations of degree $n/2$ polynomials (instead of degree $n/3$), and this then runs into the problem of not being decodable if as many as $n/2-1$ parties are malicious. To solve the decoding problem, capitalizing on preprocessing, one can devise **robust secret-sharing schemes** where the shares also contain MACs (message-authentication-codes) using secret authentication keys [50, 28, 14, 30, 29]. Of course, the MACs need to have linearity properties to be usable in the online stage of an MPC protocol.

Two main challenges remain. First, in the honest-majority setting, can the online-stage "opening" of the robust secret-shares have communication cost as low as a small constant number of field elements per opening per party, as shown by [26] for two-thirds majority. Second, how expensive is the preprocessing stage to generate the correlated randomness, i.e. the robust secret-sharing. To address the first challenge, a remarkable constant round constant communication (per opening per party) protocol was given by [29] (when amortized over $\Theta(n^2)$ simultaneous openings). This enabled [29] to give the first honest-majority preprocessing MPC where the online stage had linear communication complexity $O(n|C|)$ and $O(D)$ rounds. However, their protocol has the drawback that the circuit width must be $\Omega(n^2)$ in order to achieve linear communication complexity (in contrast to the $t < n/3$ case where the circuit width only needs to be $\Omega(n)$). They mentioned it as an open problem to achieve similar communication complexity for $\Omega(n)$-width circuits so as to match [26]. Moreover, their robust secret-sharing requires $\mathrm{O}(n^3)$ multiplications per encoding during preprocessing (i.e. the offline stage of honest-majority MPC).

1.1 Our Main Contribution

In this work, we devise a novel honest-majority robust secret-sharing scheme, and show that (1) it has a constant round constant communication cost (per opening per party) when amortized over only $\Theta(n)$ simultaneous openings, *and* (2) the dealing of such shares for a random value in a finite field $\mathbb{F}$ can itself be done using an honest-majority MPC protocol using only $O(n^2)$ multiplications between Shamir-shared $\mathbb{F}$ values. Thus, we obtain the first honest-majority preprocessing MPC where the online stage has linear communication complexity $O(n|C|)$ and $O(D)$ rounds for circuits with width $\Omega(n)$, matching the width requirement of the $t < n/3$ setting. It uses a novel application of list-decoding [52, 41] of folded Reed-Solomon codes [40]. Additionally, we have a more concretely efficient opening procedure (over [29]) when amortized over $\Theta(n^2)$ simultaneous openings, e.g. if a circuit depth has width $\Theta(n^2)$. Both of our opening procedures require four rounds, whereas [29] required seven rounds. Furthermore, [29] required all elements to be in a large extension field $\mathbb{E}$ of size at least 2^λ (for statistical soundness of the openings), whereas our robust secret-sharing scheme can have some elements come from a smaller field $\mathbb{F}$ (the field in which the shared

value lies). Additionally, our *preprocessing* cost is a factor $O(n)$ better than earlier robust secret-sharing schemes. There are other technical contributions for the preprocessing, as it is not immediately clear how the different randomness of Shamir shares, MACs and MAC-key shares are generated during the preprocessing (using say, standard honest-majority fully-secure MPC such as [39, 15]). As a technical benefit over [29], our robust secret-sharing scheme is information-theoretic private, whereas [29] is only statistically private.

Using our novel honest-majority robust secret-sharing scheme, we obtain the following theorems.

Theorem 1.1. *There exists an honest-majority fully-secure MPC protocol in the preprocessing model with online communication complexity* $32n|C|$ *elements of* $\mathbb{F}$ *and* $8n|C|$ *elements of* $\mathbb{E}$ *(for circuits over* $\mathbb{F}$ *with width* $\Omega(n)$*) and* $4D$ *rounds. The preprocessing material for this MPC protocol can be generated with* $O(n^3|C|)+\mathsf{poly}(n)$ *communication using the honest-majority MPC protocol of [39].*

Theorem 1.2. *There exists an honest-majority fully-secure MPC protocol in the preprocessing model with online communication complexity* $8n|C|$ *elements of* $\mathbb{F}$ *and* $8n|C|$ *elements of* $\mathbb{E}$ *(for circuits over* $\mathbb{F}$ *with width* $\Omega(n^2)$*) and* $4D$ *rounds. The preprocessing material for this MPC protocol can be generated with* $O(n^3|C|)+\mathsf{poly}(n)$ *communication using the honest-majority MPC protocol of [39].*

1.2 An Additional Contribution

We show that the same novel robust secret-sharing scheme can be used to boost the *correctness threshold* of offline-online BGW to honest-majority (while the privacy and robustness thresholds remain at $n/3$ corruptions). This means that if an honest party's output of the computation is not $\perp$, it is guaranteed to be correct even if the number of corruptions exceeds $n/3$ provided that there is still an honest-majority. While Theorems 1.1 and 1.2 provide honest-majority full-security, their online speed is still slower than online-BGW (see Table 1, as well as more detailed discussion below in Section 1.3). So, if we could boost the offline-online BGW to $n/2$-threshold correctness, it can have applications in secure elections and (commodity, bond market and stock market) secure auctions[2], where certification of a correct result is paramount, while still maintaining $n/3$-privacy and robustness.

We show that one can run the offline-online BGW protocol (using degree $n/3$ Shamir shares) and in parallel use the robust secret-sharing for *local* processing by the parties (i.e. no communication required), and then do a final aggregate check (for all gates) using only a constant number of openings of the robust secret-shares (i.e., using a global MAC key as in SPDZ [27]). Note that although there are only a constant number of openings, the number of robust secret-shares precomputed is same as the number of multiplication gates. Thus, it is important to lower the preprocessing cost to generate such robust sharings. We obtain the following theorem.

[2]See Supplementary Material M for a concrete example and cost of running our protocol.

Theorem 1.3. *There exists a two-thirds honest-majority fully-secure MPC protocol in the preprocessing model with the additional guarantee of correctness even against up to $n/2$ malicious corruptions. This protocol has online communication complexity $12n|C|$ elements of $\mathbb{F}$ and $10n^2$ elements of $\mathbb{E}$ (for circuits over $\mathbb{F}$ with width $\Omega(n)$). The preprocessing material for this MPC protocol can be generated with $O(n^3|C|) + \mathsf{poly}(n)$ communication using the honest-majority MPC protocol of [39].*

A natural question is whether the protocol of Thm. 1.3 can be extended to also *boost the robustness threshold* to $n/2$. It turns out that this can be easily done by having the parties simply run the online stage of our honest-majority MPC protocols (Thms. 1.1 or 1.2) using the same preprocessing material in the event that the initial output was $\perp$. Such a protocol is called fast-track [36] in the sense that if everything is fine (in our case less than $n/3$ corrupt), the online stage runs quickly, and the slower online stage of Thms. 1.1 or 1.2 is only run if $> n/3$ corruptions are detected. Note that this fast-track protocol has no additional offline cost over the protocol of Thm. 1.3 since it uses the same preprocessing material. Due to lack of space, we describe this extension in Supplementary Material I.

1.3 On the $t < n/3$ vs. $t < n/2$ Settings

Although the online stage of our honest-majority protocols in Theorems 1.1 and 1.2 asymptotically match the online stage of offline-online BGW in the $t < n/3$ setting, they are concretely less efficient for several reasons.

- **Non-responsive Protocol:** The honest-majority protocols are non-responsive, meaning that they must be run over a synchronous network with an upper bound Δ on the network delay, and all parties must wait Δ time in each round, even if the actual network delay is significantly less than Δ. This is because it does batch verification at each depth, which requires commitments. In contrast, the online stage of protocols in the $t < n/3$ setting [12, 11] is responsive [48], meaning that parties that have received messages from all other parties for a round can proceed to the next round without waiting a fixed amount of time Δ. Since the actual network delay is often significantly less than the upper bound Δ, responsive protocols can run and finish in a fraction of the total time compared to non-responsive protocols.
- **Hidden Constants:** The asymptotic communication complexities are measured in terms of field elements sent by the parties, but in the $t < n/2$ setting, many of the field elements communicated must come from an extension field $\mathbb{E}$ with $|\mathbb{E}| > 2^\lambda$, where λ is the statistical security parameter, even if the computation is over a smaller field $\mathbb{F}$. This is due to the statistical security of the MACs used in the robust secret-sharing schemes. In contrast, in the $t < n/3$ setting, the field elements can all come from $\mathbb{F}$.

Thus, for applications requiring a very fast online stage, using a $t < n/3$ protocol may be necessary for concrete efficiency reasons even though it has weaker security guarantees. These scenarios motivate Thm. 1.3.

Protocol	Corruption Thresholds	Communication	Field Size	Circuit Width	Rounds	Responsive Rounds
BGW-style [26, 11]	$(n/3, n/3, n/3)$	$12n\|C\| * \log \mathbb{F}$	$> n$	$\Omega(n)$	$2D$	✓
SPDZ† [27]	$(n, 1, n)$	$2n\|C\| * \log \mathbb{F}$ $+O(n^3) * \log \mathbb{E}$	any $> 2^\lambda$	$\Omega(1)$	$2D + O(1)$	✓*
EF23 [29]	$(n/2, n/2, n/2)$	$18n\|C\| * \log \mathbb{E}$	$> 2^\lambda$	$\Omega(n^2)$	$7D$	✗
This Work (Theorem 1.1)	$(n/2, n/2, n/2)$	$32n\|C\| * \log \mathbb{F}$ $+ 8n\|C\| * \log \mathbb{E}$	$> 3n$ $> 2^\lambda$	$\Omega(n)$	$4D$	✗
This Work (Theorem 1.2)	$(n/2, n/2, n/2)$	$8n\|C\| * \log \mathbb{F}$ $+ 8n\|C\| * \log \mathbb{E}$	$> n$ $> 2^\lambda$	$\Omega(n^2)$	$4D$	✗
This Work (Theorem 1.3)	$(n/3, n/3, n/2)$	$12n\|C\| * \log \mathbb{F}+$ 1 $\mathsf{Byz}^{**} + 10n^2 * \log \mathbb{E}$	$> n$ $> 2^\lambda$	$\Omega(n)$	$2D + O(1)$	✓*

Table 1: Comparison of the online phase of information-theoretic MPC protocols with $O(D)$ rounds in the preprocessing model. The corruption thresholds are of the form (t_p, t_r, t_c), the thresholds for privacy, robustness, and correctness, respectively. Byz denotes the communication of an honest-majority Byzantine consensus which can be run in expected constant rounds assuming the servers have certified public-keys [45]. The double asterisk on Byz signifies that if no party was erroneous then the Byzantine consensus can be stopped short asynchronously (see last step of Protocol $\Pi_{\mathsf{ONLINE\text{-}X\text{-}SFE}[\mathbb{F},f,n,m]}$). In other words, Byz has an impact on total communication only if more than $n/3$ parties acted maliciously. ✓* means the protocol is responsive until the attestation phase at the end (the final $O(1)$ rounds).
† The reported results for SPDZ count an ideal broadcast as unit cost.

1.4 Related Work

We have already discussed several relevant related works above, but here we give a more thorough review of the related literature in honest majority MPC with robustness (not necessarily with pre-processing). In the $t < n/3$ setting, [12] and concurrently [20] showed that robust protocols with perfect security are possible. [42] introduced the player elimination framework, and [11, 38] achieved linear communication complexity in this setting, but required $\Omega(D + n)$ rounds when considering both offline and online phases. [2, 1] showed how to obtain $O(D)$ rounds while maintaining good communication complexity. In the $t < n/2$ setting, the works [10, 13, 39, 22, 29, 4] have focused on improving the communication and round complexity. In particular, [4] achieves honest-majority robust MPC with $O(D)$ rounds while improving on prior works in terms of communication complexity when considering both the offline and online phases. In the $t < n/3$ setting, packed secret sharing yields protocols with communication sublinear in the circuit size under certain width assumptions [25]. In the honest-majority setting, [44] gave compilers that upgrade secure-with-abort protocols to

guaranteed output delivery, though the resulting round complexity can depend on n due to the need to identify and eliminate corrupt parties.

A separate line of work studies protocols that provide different security guarantees under different corruption thresholds. Ishai et al. [43] introduced the "best of both worlds" paradigm, achieving security that adapts to the actual corruption pattern. Related directions include extensions of SPDZ with fairness [53] and recent work on "best-in-the-multiverse" style MPC [3] that explores similar trade-offs across thresholds.

On the practical side, concretely efficient protocols have been designed for small party counts, including three-party computation with security-with-abort [32, 21]. Robust secret sharing and VSS with efficient reconstruction have been studied extensively [24]; our construction relies on a variant of such schemes tailored to the preprocessing model.

The works of [48, 49] introduce the notion of responsiveness for protocols in the context of blockchains, and [47] discusses responsive protocols in the context of Byzantine agreement.

2 Technical Overview

2.1 Our Novel Robust Secret-Sharing Scheme

Previously known robust secret-sharing schemes [28, 14, 30, 29] can be thought of as bivariate codes first used in MPC in BGW [12] (see also [5]). In a bivariate code, to encode $a \in \mathbb{F}$, instead of choosing a random univariate polynomial $f(X)$ of degree t with constant term set to a, a random bivariate polynomial $\mathcal{U}(X,Y)$ of degree t (in each variable) is chosen with constant term a. Each component of the codeword (each party's share) consists of two degree t univariate polynomials $u_i(X)$ and $v_i(Y)$ which are respectively $\mathcal{U}(X,\zeta_i)$ and $\mathcal{U}(\zeta_i,Y)$. Here ζ_i are public evaluation points associated with party P_i. Unfortunately, even though the *error-correction capability* of this code is as high as t for $t = n/2$ (see e.g. [5, Lemma 5.3]), there is no polynomial-time algorithm known to perform such error-correction[3].

To obtain (efficient) robustness for $t = \lfloor (n-1)/2 \rfloor$, the robust secret-sharing schemes employ secret evaluation points, say $\theta_1, \ldots, \theta_n$ [28], with one evaluation point being secretly given to each server (during preprocessing). More precisely, while $u_i(X) = \mathcal{U}(X,\zeta_i)$, just as in BGW and employing public evaluation points, the other polynomial $v_i(Y) = \mathcal{U}(\theta_i,Y)$ employs secret evaluation points. During an "opening", a party P_i's shares are authenticated by P_j as follows: party P_i sends its univariate polynomial $u_i(X) = \mathcal{U}(X,\zeta_i)$. Since party P_j has its secret evaluation point θ_j, as well as $v_j(Y) = \mathcal{U}(\theta_j,Y)$, it checks if $v_j(\zeta_i) = u_i(\theta_j)$. It is not difficult to show that the probability of an incorrect $u_i(X)$ passing this test is negligible. We remark here that since the secret evaluations, i.e. $v_j(Y)$,

[3] The error-correction for this code (for $t = n/2$) requires finding cliques, which is NP-hard.

are never revealed by party P_j, the secret evaluation points can be reused for multiple sharings. Further, the above verification property is linear.

One big drawback of these known robust secret-sharing schemes is that the preprocessing cost (the offline stage of an MPC protocol) to obtain these shares for each party is quite high. This is because they require $O(n^3)$ multiplications (of field elements) to calculate the various shares, and in particular $\mathcal{U}(\theta_i, Y)$. Since θ_i is secret to each party P_i, and the coefficients of $\mathcal{U}(X, Y)$ are also random and secret, to generate the preprocessing, these values are presumably Shamir secret-shared with shares held by the various parties. The values $\{\mathcal{U}(\theta_i, Y)\}_{i\in[n]}$ can be seen as matrix multiplication of an $n \times (t+1)$ Vandermonde matrix (with secret evaluation points θ_i) and a $(t+1) \times (t+1)$ matrix representing $\mathcal{U}(X, Y)$.

In this work, we depart from the bivariate-code methodology to obtain robust secret-sharings, and instead use **affine MAC**s using secret keys θ_i. Since the MACs are never revealed even during "openings", a single-element key θ_i suffices for each party P_i (and is reused over all encodings). More precisely, $n+1$ random and independent degree-t polynomials are chosen, $F_0(X), F_1(X), ..., F_n(X)$, with the restriction that $F_0(0) = a$, where a is the value being encoded. Each party P_i is given $F_\ell(\zeta_i)$ (for all $\ell \in [0..n]$). Each party P_j is also given precomputed affine MACs, namely $\mathsf{mac}_{j,i} \stackrel{\text{def}}{=} F_0(\zeta_i) + \theta_j * F_j(\zeta_i)$. In other words, the first polynomial $F_0(X)$ (the constant term of which has the value a) is authenticated using polynomial $F_j(X)$ as *auxiliary information* for target party P_j. See Lemma 3.2 for a precise statement and authentication claim. The reason the polynomials $F_j(X)$ are degree t and only their evaluations (at ζ_i) are given to parties P_i is because this is needed during the communication-efficient "opening" a la [26], as we will soon explain. It is not difficult to see that now the preprocessing cost is only $O(n^2)$ multiplications (of Shamir secret-shared values), as each (secret) θ_j is only multiplied with (secret polynomial) $F_j(X)$.

Challenges of Efficient Robust Opening. At first glance, it would seem that the communication cost of opening has also gone down, as each party P_i now just needs to send $F_0(\zeta_i)$ and $F_j(\zeta_i)$ to party P_j (as opposed to a whole polynomial $u_i(X)$ in earlier robust secret-sharing openings). However, this is still a total of $2n^2$ total communication over all parties for a single opening. Our aim is to bring it down to $O(n)$ total communication per opening (amortized). We can try to employ the [26] two-round amortized opening (to simultaneously perform $n/2$ openings). In that scheme, which works for less than $(n-t)/2$ errors (for a degree t Shamir secret-sharing), $t+1$ independent openings are simultaneously performed, say of $[a^{(1)}], [a^{(2)}]$, ..., $[a^{(t+1)}]$. Recall, in Shamir secret-sharing scheme, there are no MAC tags, and decoding is completely based on Reed-Solomon error-decoding. Thus, each party P_i can take its shares of the various sharings, say, $[a^{(1)}]_i, [a^{(2)}]_i$, ..., $[a^{(t+1)}]_i$, and expand them using a degree t Shamir secret-sharing (employing public evaluation points $\{\zeta_j\}_j$) to a vector of n elements. P_i then sends P_j the j-th element from this expansion. By linearity of Shamir sharing, each party P_j collectively receives a Shamir secret sharing of degree t, which encodes (in the constant term) $\sum_{k\in[t+1]} \zeta_j^k \cdot a^{(k)}$. With at most $(n-t)/2$ erroneous parties, P_j can Berlekamp-Welch decode (see

e.g. [34]) to obtain this constant term. In the next phase, each party P_j sends to all other parties this decoded quantity. Thus, in this phase, each party P_ℓ receives a degree-t Reed-Solomon codeword encoding the polynomial that has coefficients $a^{(1)}, a^{(2)}, ..., a^{(t+1)}$. Again, under the corruption threshold, each party can Berlekamp-Welch decode these values.

However, as opposed to [26], which required no secret-key checks for decoding since they are in the $t < n/3$ setting, the second round of [26] for our robust secret-sharing becomes difficult to decode in the honest-majority setting as the authentication material is not easy to provide. One can give a protocol to authenticate the material in the second round, but the communication complexity is still high, i.e. a total of n^3 elements communicated for $n/2$ simultaneous openings. However, since this cost is due to the auxiliary information sent for the MAC checks, these can be further aggregated as a batch verification using powers of a freshly opened random value. This batch verification idea was used in [29] to verify n batches of $n/2$ shares in total communication $\Theta(n^3)$. However, as previously mentioned, this approach had the downside of requiring $\Theta(n^2)$ simultaneous openings. Our robust secret-sharing scheme also supports this batched-opening approach for the same $n(t+1) = \Theta(n^2)$ simultaneous openings, but is more concretely efficient (since it can send many elements over the smaller field $\mathbb{F}$) and uses four communication rounds, compared with seven for [29]. The three round saving comes from directly authenticating the first stage messages, thereby avoiding the additional challenge-opening and subsequent checking round of [29], and from sending the final auxiliary verification data directly from P_i to P_ℓ, rather than relaying it through an intermediary P_j.

2.2 Efficient $\Theta(n)$ Simultaneous Openings via List Decoding

The key bottleneck in efficiently opening $\Theta(n)$ shares simultaneously via the above approach is that the authentication material in the second round requires n^3 communication. So, what if we simply do not authenticate it? At first glance, this seems like a completely nonsensical proposition, since we are well beyond the error decoding bound. However, we are not restricted to using the standard Reed-Solomon outer code used in [26, 29] to encode the first round messages (the sharings of sharings) sent by the parties and perhaps a different outer code will enable decoding of the second round messages without authentication. This is also not possible (without the size of the outer code symbols being at least the size of $\Omega(n)$ base field elements) again due to the error decoding bound. However, we are able to show (see Lemma 5.1) that if we use a 3-folded Reed-Solomon code, then there are at most 2 possible decodings! Using the Guruswami-Sudan list decoder [41] allows the parties to efficiently decode and obtain this list of at most two polynomials. The final challenge is for the parties to efficiently identify which of the two polynomials is the correct one. By choosing a challenge point on which these polynomials disagree, the parties can efficiently obtain authenticated sharings of the correct polynomial evaluation at the challenge point. This enables them to decode the correct polynomial evaluation and determine which of the two polynomials in their decoded list is correct. To further reduce the

concrete communication cost (in terms of the base field $\mathbb{F}$ and the larger extension field $\mathbb{E}$), we note that the 3 elements of $\mathbb{F}$ (the 3-folded encoding) sent in the first round can be authenticated with a single element of $\mathbb{E}$ by aggregating the authenticators. Moreover, while the challenge point would typically need to be in $\mathbb{E}$ (to obtain a negligible probability of failure), the challenge point in our opening procedure can be in $\mathbb{F}$ since each party can easily determine a point where the two polynomials in their list are not equal.

2.3 Enhancing the Correctness of Offline-Online BGW

BGW-style MPC protocols operate with the inputs/intermediate values secret-shared amongst the parties. When using Beaver tuples generated in the pre-processing phase, the online phase operates by consuming a Beaver tuple per multiplication gate and having the parties open blinded shares. When $t < n/3$, malicious behavior can easily be error corrected, *guaranteeing* that honest parties receive the correct openings and eventually the correct output(s). However, in the $t < n/2$ case, malicious behavior is not easily error corrected, unless one uses robust codes as we did in Theorems 1.1 and 1.2. The other approach is to use the observation that (most) semi-honest MPC protocols only have "additive" malicious attacks [27, 35]. One can then obtain maliciously secure protocols by using a global MAC-key as in SPDZ [27], with BGW-style online efficiency (also see [35, 46, 21]). However, these only attain security-with-abort as these do not have error-correction capabilities.

Our approach is to employ both these ideas in a novel way to get fast MPC that has $n/3$ privacy and robustness thresholds, but honest-majority correctness. We will employ precomputed Beaver tuples as well as authenticated Beaver tuples. The main idea is to use an $n/3$-privacy threshold secret-sharing scheme for the basic Beaver tuples, and an $n/2$-robust secret-sharing scheme for the authenticated tuples. During the execution of the protocol, the authenticators are *locally* updated but not opened. Then, at the end of the protocol, there is an attestation phase, where these authenticators are aggregated and then opened, attesting to the entire computation with a low communication/round overhead. This is similar to the approach employed by SPDZ; however, since we are using sharings at two different thresholds, getting the underlying shares and the authenticated shares to work together is a challenge.

The first observation is that if more than $n/3$ servers are corrupt (though still less than $n/2$), while privacy of the clients' inputs are lost (as these were secret shared using an $n/3$-threshold scheme), the authentication key is still private (as the authenticators were secret shared using an $n/2$-threshold scheme), and maybe this allows us to show that attacks are "additive only" with malicious additions independent of the authentication key. Indeed, the "additive only" assumption does hold in a simulation model where the ideal world simulator is given access to the inputs and Beaver tuples (everything shared at threshold $n/3$) when $t \geq n/3$. So, the additive errors introduced in this case may depend on the inputs, but they are still independent of the authentication key α. The challenge is to prove the SPDZ-style attestation at the end is secure.

We are in a unique setting where we want robustness as well (at least for $t < n/3$), so we need to use a secret-sharing scheme (which is also robust), whereas prior works that only offered security with abort [46, 21] in the honest-majority setting didn't need to use a robust secret-sharing scheme. However, these schemes did use a Shamir secret-sharing scheme as they were not using precomputed Beaver multiplication-tuples (as opposed to SPDZ). Our scheme uses Beaver multiplication-tuples but also uses robust secret-sharing schemes rather than just additive secret-sharing (SPDZ). Regardless, it is worth comparing our technique to [21], which also performs a SPDZ-style aggregation of Shamir secret-shares. In [21], they use a different random value as the linear combiners for the aggregation. This results in having to open a different random value for every multiplication gate in the circuit, rather than a single value. However, this greatly simplifies the proof because they can argue that one of the combiners (say x_i) is independent of all other values, and thus, the probability of equality is $1/|\mathbb{F}|$. If they had used a single x for aggregation, this would be false, since the equation on the other side would contain other powers of x. However, we are using a robust secret-sharing scheme for the aggregation and opening a random value for every multiplication gate in the circuit would be prohibitively expensive. [21] points out that one can open a single random value and then use a PRG to generate the random combiners for the aggregation, but then we no longer have information-theoretic security.

We have a more involved proof that is able to handle the challenges of the honest-majority setting and use a single x for aggregation. At a high level, we analyze the errors introduced by the adversary in the adversary's own shares and the honest parties' shares separately. The errors introduced in the adversary's shares are independent of α, whereas the errors introduced in the honest parties' shares may depend on α, but, crucially, are of a specific form $c \cdot \alpha$ with c independent of α. We are able to conclude that for the attestation to pass, an equality of the form $W = \alpha \cdot V$ must hold for W, V independent of α. Intuitively, one can think of W as the accumulation of errors from the adversary's shares and V as the accumulation of errors from the honest parties' shares with α factored out (here, we crucially use that the errors introduced in the honest parties' shares are of the form $c \cdot \alpha$). From this, we can conclude that $W = V = 0$ with overwhelming probability. At this point, we are able to apply Schwartz-Zippel to V (an aggregation using a single x) because while the adversary may have introduced errors into the individual gate checks of the honest parties, the honest parties will honestly aggregate these terms and, therefore, we are guaranteed that V is a polynomial in X. This allows us to conclude that all the individual gate checks are themselves zero, and then, by an inductive argument, we conclude that all errors were in fact zero.

3 Preliminaries

Our MPC protocols operate over secure and authenticated point-to-point channels. We assume a synchronous model, where messages sent by honest parties

to other honest parties are guaranteed to arrive within Δ time steps, where Δ is a predetermined upper bound on the network latency. We prove our results against a malicious adversary with static corruptions.

Responsive Protocols. We say that a protocol (or a round of a protocol) is *responsive* if a party that receives messages in a round of a protocol from all parties (from which it expects a message) in time $\delta < \Delta$ can proceed to the next round. This is in contrast to non-responsive protocols, where a party must wait Δ time before proceeding to the next round, even if it has already received all the expected messages for a round.

3.1 Secret Sharing Preliminaries

Let $\mathbb{F}$ be a finite field over which the arithmetic circuit is defined. Let $\zeta_1, \zeta_2, ..., \zeta_n$ be distinct non-zero elements of $\mathbb{F}$. We will assume $|\mathbb{F}| \geq n+1$ so that these distinct non-zero elements can be chosen.

A (d, n)-Shamir secret share of an element $a \in \mathbb{F}$ is the list of n elements $([a]_i = f(\zeta_i))_{i \in [n]}$ where $f(X)$ is a degree $d-1$ polynomial defined over $\mathbb{F}$ with the constant term of $f(X)$ set to a. The full collection of n shares is denoted by ${}^{d}[a]$, or just $[a]$ when d is clear from context. The elements $\zeta_1, \zeta_2, ..., \zeta_n$ are referred to as the **evaluation points** of the Shamir secret-sharing scheme (as well as the related Reed-Solomon code). Using the above evaluation points, define the **Vandermonde matrix** $\mathbf{V}^{n \times d}$, with $\mathbf{V}_{i,j} = \zeta_i^{j-1}$ (for $i \in [n]$ and $j \in [d]$).

Then, ${}^{d}[a] = \mathbf{V}^{n \times d} \cdot \mathbf{f}$ where $\mathbf{f}$ is the vector of coefficients of $f(X)$ with constant term a, i.e. $\mathbf{f}_0 = a$. It is well-known that any d rows of the above matrix form an invertible matrix. Hence, any d shares out of the n shares can be used to (efficiently) determine a. Note that Shamir secret-sharing is linear in that $[a] + [b] = [a + b]$, and $\lambda \cdot [a] = [\lambda \cdot a]$ for any $\lambda \in \mathbb{F}$.

When we say that ${}^{d}[a]$ is a **fresh random** Shamir secret-sharing of a, we mean that the vector $\mathbf{f}$ was chosen uniformly at random from $\mathbb{F}^d$ conditioned on $\mathbf{f}_0 = a$. In this work, we will mostly use two types of Shamir secret-shares, one with $d = n/3$ and another with $d = n/2$. When sharings with both thresholds are in use, we will use the notation $[a]$ for ${}^{n/3}[a]$ and $\langle a \rangle$ for ${}^{n/2}[a]$.

The robust secret-sharing schemes we use consist of Shamir secret-shares augmented with additional information. We will use the notation $[\![a]\!]$ to refer to the augmented shares of a. If the augmentation is of ${}^{n/2}[a]$, i.e. $\langle a \rangle$, then we will use the notation $\langle\!\langle a \rangle\!\rangle$ to refer to the augmented shares. These augmented shares make use of an extension field $\mathbb{E}$ of $\mathbb{F}$. If the base field $\mathbb{F} = \mathbb{E}$ (that is, the secret $a \in \mathbb{E}$ and the Shamir secret sharing is done over $\mathbb{E}$), we will use the notation $[\![a]\!]^{\mathbb{E}}$ (resp., $\langle\!\langle a \rangle\!\rangle^{\mathbb{E}}$). Our robust secret-sharing scheme uses secret evaluation points $\theta_1, \cdots, \theta_n$ that are sampled during share generation. Sometimes, we will require these secret evaluation points to be the same across multiple sharings. In this case, we will explicitly mention this and use the notation $\vec{\theta}$ to refer to the vector of secret evaluation points $\theta_1, \cdots, \theta_n$ that are common across sharings.

3.2 Opening Sharings for $t < n/3$

We overview the procedure from [26] for n parties $P_1, \cdots, P_n$ to robustly open consistent degree $t < n/3$ Shamir secret shares in the presence of a malicious adversary controlling at most t parties. Each party P_i possesses share x_i of some secret x. We first describe an opening procedure that relies on a chosen party P^*.

Open$(P^*, t, [x])$

1. Each party P_i sends its share x_i of $[x]$ to P^*.
2. P^* runs Berlekamp-Welch on the received shares to recover x and sends x to all parties.

Since $t < n/3$, it follows that there are at least $2t+1$ correct shares and at most t incorrect shares given to P^*. Thus, Berlekamp-Welch [34] will correctly decode x. Of course, the above opening procedure has the issue that it requires P^* to be honest. To handle this issue, [26] give an opening procedure, that simultaneously opens $\ell = n - (2t + 1)$ sharings by first expanding them into n sharings using a linear error correcting code. Each party is tasked with opening one sharing and then the dishonest openings can be corrected. The opening procedure is given below.

OpenRobust$(t, [x^{(1)}], \cdots, [x^{(\ell)}])$

1. Each party P_i applies an $n \times \ell$ Vandermonde matrix to its shares $(x_i^{(1)}, \cdots, x_i^{(\ell)})$ to obtain shares $y_i^{(1)}, \cdots, y_i^{(n)}$.
2. For each $i \in [n]$, run (in parallel) $\mathsf{Open}(P_i, t, [y^{(i)}])$.
3. All parties run Berlekamp-Welch on received values $y^{(1)}, \cdots, y^{(n)}$ to obtain $x^{(1)}, \cdots, x^{(\ell)}$.

Since affine combinations of degree t Shamir sharings produce degree t Shamir sharings, the $y_i^{(j)}$'s are correct sharings of $y^{(j)}$. Since there are at most $t < n/3$ dishonest parties, at most t of the received $y^{(j)}$'s will be incorrect, and so Berlekamp-Welch will correctly decode $x^{(1)}, \cdots, x^{(\ell)}$.

3.3 Random Affine Functions

While affine functions are commonly seen as universal hash functions [19] (also see e.g. [33]), we will use two other dual properties of random affine functions. Let $\mathbb{F}$ be a finite field.

Lemma 3.1. *For any non-zero $s \in \mathbb{F}$, the distribution of $a * s + b$, with a chosen randomly from $\mathbb{F}$, is independent of b sampled from any distribution with support in $\mathbb{F}$.*

Proof. The distribution of $a * s$ is random and uniform in $\mathbb{F}$ (when $s \neq 0$, as $\mathbb{F}^*$ is a multiplicative group), and the lemma follows as $\mathbb{F}$ is an additive group.

Lemma 3.2. *Given any $a, b, a', b' \in \mathbb{F}$, such that $(a, b) \neq (a', b')$*

$$\Pr_{s \xleftarrow{\$} \mathbb{F} \setminus \{0\}} [b' * s + a' = b * s + a] \leq 1/(|\mathbb{F}| - 1)$$

Proof. If $b = b'$ then from given hypothesis that $(a, b) \neq (a', b')$, the probability of the above is zero. If $b \neq b'$, we must have $s = (a - a')/(b' - b)$, the probability of which is just $1/(|\mathbb{F}| - 1)$.

The above lemma is used as follows. For any **mac-tag** on a with **auxiliary information** b (and **secret random key** s), no other a' with any b' is likely to compute to the same mac-tag. Thus, if a party knows both a and b, but does not know the mac-tag nor the key s, then the probability some other a' and b' computes the same mac-tag is negligible.[4]
We now state the usual 2-universal property [33] of affine random functions. This property is used when even the mac-tag is known. But, now two secret keys (s_1, s_2) are required. This 2-universal (MAC) property is used in [27] as well as our enhanced-$n/3$ scheme.

Lemma 3.3. *For all $a, a' \in \mathbb{F}$, $a \neq a'$, for all $\gamma, \gamma' \in \mathbb{F}$,*

$$\Pr_{s_1, s_2 \xleftarrow{\$} \mathbb{F}} [a * s_1 + s_2 = \gamma \mid a' * s_1 + s_2 = \gamma'] \leq 1/|\mathbb{F}|$$

4 A Novel Robust Secret-Sharing Scheme

In this section, we describe a novel robust secret-sharing scheme. Assume that there are n parties P_j ($j \in [n]$). Let $\mathbb{F}$ be the finite field (of size at least $n + 1$) over which the underlying function is defined. Thus, the values to be encoded lie in $\mathbb{F}$. Let $\mathbb{E}$ be the extension field of $\mathbb{F}$ which has at least 2^λ elements (where λ is a security parameter) and let $\{\zeta_j\}_{j \in [n]}$ be n fixed (and public) distinct elements of $\mathbb{F}$. When a subset S of $\{\zeta_1, ..., \zeta_n\}$ is specified, we write $\mathbf{V}^{S,d}$ for the Vandermonde matrix with $\mathbf{V}^{S,d}_{i,j} = S_i^{j-1}$ (for $i \in [|S|]$ and $j \in [d]$).

Instead of the bivariate-code robust secret-sharing used by [29], namely a bivariate polynomial $C(X, Y)$ of degree t in each variable, our robust sharing scheme will use an affine random function (see Section 3.3) in one dimension and a degree t polynomial encoding in the other dimension. The sharing scheme is described in Figure 1.

Note that from party P_j's share, P_j can compute

$$\mathsf{mac}_{j,*} \stackrel{\text{def}}{=} F_0(0) + \theta_j * F_j(0) \tag{1}$$

[4] One can also view this authentication as the usual 2-universal hash (see Lemma 3.3) by treating the **mac-tag** as an *ephemeral secret key* (as it is not disclosed as part of the message), and treating the auxiliary information b as 2-universal hash. To see this, note the secret keys are now a pair $(s, b * s + a)$, which can equivalently be considered as secret key pair $(-s^{-1}, (b * s + a) * s^{-1})$, and then b becomes a 2-universal hash of a under this key, viz. $b = a * (-s^{-1}) + (b * s + a) * s^{-1}$.

A Robust Secret-Sharing Scheme $[\![\cdot]\!]^{\mathbb{F},\mathbb{E},n}$

- Assume $\mathbb{E}$ is an extension field of $\mathbb{F}$. Let $t = \lceil n/2 \rceil - 1$. On input $a \in \mathbb{F}$, do the following:
  1. Sample a random degree-t polynomial $F_0(X)$ over $\mathbb{F}$, conditioned on constant term set to $a \in \mathbb{F}$, *the value being shared*.
  2. Sample n additional random and independent degree-t polynomials over $\mathbb{E}$, $F_1(X), ..., F_n(X)$. We will write this vector of polynomials (i.e. $(F_0(X), ..., F_n(X))$) as $\mathbf{F}(X)$ (note, it includes $F_0(X)$).
  3. **Each party P_j gets $\mathbf{F}(\zeta_j)$**, which are the $(n+1)$ evaluations of $\mathbf{F}(X)$ at ζ_j. Note that $\{F_0(\zeta_j)\}_{j\in[n]}$ is typically written as $[a]$, i.e the degree-t Shamir secret-sharing of a. Further note that each value $F_0(\zeta_j)$ is in $\mathbb{F}$, and all other $F_i(\zeta_j)$ are in $\mathbb{E}$. We will refer to the values $F_i(\zeta_j)$ (for $i \in [1..n]$) as **auxiliary information.**
  4. Each party P_j also gets a **verification key** $\theta_j \in \mathbb{E}$ (which is chosen randomly and independently *conditioned on being non-zero*). The same secret verification key can be reused over many different sharings.
  5. Each party P_j gets $n-1$ **affine mac-tags**, namely $\mathsf{mac}_{j,i} = F_0(\zeta_i) + \theta_j * F_j(\zeta_i)$ (for $i \in [1..n], i \neq j$). Note, the $\mathsf{mac}_{j,i}$'s lie in the extension field $\mathbb{E}$.
- Thus, Party P_j's share of $[\![a]\!]$ consists of (i) $\mathbf{F}(\zeta_j)$, (ii) verification key θ_j, and (iii) $n-1$ mac-tags $\{\mathsf{mac}_{j,i}\}_{i\in[n],i\neq j}$.

Fig. 1: A Novel Robust Secret-Sharing Scheme

from any $(t+1)$-size subset of macs (e.g., $\{\mathsf{mac}_{j,i}\}_{i\in[t+1]}$), by applying the inverse of the Vandermonde matrix $\mathbf{V}^{S,t+1}$ to the vector $[\mathsf{mac}_{j,i}]_{i\in[t+1]}$, where S is the set $\{\zeta_1, ..., \zeta_{t+1}\}$. This fact will be useful during our simultaneous robust opening protocol. Our robust secret-sharing scheme also satisfies the following linearity property.

Lemma 4.1 (Linearity). *Given a secret-sharing $[\![a^{(1)}]\!]$ of $a^{(1)}$ and another secret-sharing $[\![a^{(2)}]\!]$ of $a^{(2)}$ (both under the same keys $\{\theta_j\}$), for any $c_1, c_2 \in \mathbb{F}$, a valid secret-sharing of $c_1 \cdot a^{(1)} + c_2 \cdot a^{(2)}$ is obtained by taking the same linear combinations (i.e. using c_1 and c_2) of all the material of the two sharings (except θ_j).*

Proof. For all i, j, we (easily) check that $c_1 \cdot \mathsf{mac}^{(1)}_{j,i} + c_2 \cdot \mathsf{mac}^{(2)}_{j,i}$ is same as

$$(c_1 F_0^{(1)}(\zeta_i) + c_2 F_0^{(2)}(\zeta_i)) + \theta_j * (c_1 * F_j^{(1)}(\zeta_i) + c_2 * F_j^{(2)}(\zeta_i)).$$

4.1 Privacy: View of The Minority Adversary in Any Encoding.

Theorem 4.2. [Privacy] *For any $a \in \mathbb{F}$, the view of the adversary that has shares of t out of n parties of a robust encoding $[\![a]\!]^{\mathbb{F},\mathbb{E},n}$ is independent of a.*

The proof of this theorem is in Supplementary Material A.

4.2 Simple Robust (Single) Opening

We show here that the robust secret-sharing scheme above allows a simple one-round opening under the honest-majority assumption. However, this opening is not communication light, and we will describe a more efficient scheme in the next subsection. The simple scheme requires each party to send two field elements to every other party. Over all parties, this amounts to $2 * n^2$ field elements communication to open a single element. We remark that earlier robust schemes (such as [29]) require $n^3/2$ total communication to open a single element.

Protocol $\Pi_{\mathsf{RSS}}.\mathtt{OpenSound}$ for $[\![a]\!]^{\mathbb{F},\mathbb{E},n}$

Input: Each party P_i has its share of $[\![a]\!]$. Thus, party P_i has $F_0(\zeta_i)$ with $F_0(0) = a$. Recall, $F_0(\zeta_i)$ can also be seen as share of P_i in Shamir secret-sharing ${}^t[a]$. Party P_i also has θ_i, $\{F_j(\zeta_i)\}_{j\in[n]}$, and $\{\mathsf{mac}_{i,j} = F_0(\zeta_j) + \theta_i * F_j(\zeta_i)\}_{j\in[n]}$,

Open-Start: Each party P_i sends to each party P_j, $F_0(\zeta_i)$ and $F_j(\zeta_i)$.

Verification and Reconstruction: Each party P_i receives from P_j two elements, say $b_0^{(j)}$ and $b_j^{(j)}$. If $b_0^{(j)} + \theta_i * b_j^{(j)} \neq \mathsf{mac}_{i,j}$, then reject P_j, otherwise accept P_j and let $b_0^{(j)}$ be considered as j-th share of a degree-t Shamir secret-sharing. If at least t parties P_j pass the test, then P_i uses any valid t shares and its own share $F_0(\zeta_i)$ to open this degree-t Shamir secret-sharings to get the constant term of the degree t polynomial, say $a^{(i)}$. Output $a^{(i)}$. If less than t parties pass the test, then output $\perp$.

Fig. 2: Simple Robust (Single) Opening

The simple protocol $\Pi_{\mathsf{RSS}}.\mathtt{OpenSound}$ is described in Fig. 2. By Lemma 3.2, each incorrect party P_j can pass with probability at most $1/(|\mathbb{E}|-1)$. Thus, by union bound, P_ℓ will catch any of the potentially (at most) $t = n/2-1$ incorrect parties with probability at least $1 - t/(|\mathbb{E}|-1)$. At this point, it can use any of the $t+1$ correct parties to decode $F_0(X)$ (by inverting the corresponding Vandermonde matrix), and thus also obtain $F_0(0)$ which is the a that was encoded. This proves the following lemma.

Lemma 4.3. *In the protocol $\Pi_{\mathsf{RSS}}.\mathtt{OpenSound}$ on input $[\![a]\!]$, if at least $t+1$ parties are honest, then each honest party outputs a, except with probability upper bounded by $t/(|\mathbb{E}|-1)$.*

We have a further aggregation property using the Schwartz-Zippel lemma.

Lemma 4.4. *For any $m = poly(n)$, and arbitrary values $\{r^{(\ell)}\}_{\ell\in[m]}$ and $\{s^{(\ell)}\}_{\ell\in[m]}$ in field $\mathbb{E}$, given their $[\![\cdot]\!]$ shares, let x be a random and independent value in $\mathbb{E}$. Then, if the shares of party i of $\sum_{\ell\in[m]} x^\ell \cdot [\![r^{(\ell)}]\!]$ are accepted by j as shares of $\sum_{\ell\in[m]} x^\ell \cdot [\![s^{(\ell)}]\!]$ with non-negligible probability in λ, then for all $\ell \in [m]$, shares of party i of $[\![r^{(\ell)}]\!]$ will be accepted by party j as shares of $[\![s^{(\ell)}]\!]$ with probability one.*

Proof. For party j to accept these shares, it must be the case that the precomputed MAC that P_j has on $\{s^{(\ell)}\}$ is $\sum_{\ell\in[m]} x^\ell \cdot \mathsf{mac}_{j,i}^{(s,\ell)}$, which by definition is $\sum_{\ell\in[m]} x^\ell \cdot \left(F_0^{(s,\ell)}(\zeta_i) + \theta_j * F_j^{(s,\ell)}(\zeta_i)\right)$. The MAC check by P_j (using its θ_j) compares this to $\sum_{\ell\in[m]} x^\ell \cdot \left(F_0^{(r,\ell)}(\zeta_i) + \theta_j * F_j^{(r,\ell)}(\zeta_i)\right)$. So, for this check to pass it must be that

$$\sum_{\ell\in[m]} x^\ell \cdot \left(F_0^{(s,\ell)}(\zeta_i) + \theta_j * F_j^{(s,\ell)}(\zeta_i)\right) = \sum_{\ell\in[m]} x^\ell \cdot \left(F_0^{(r,\ell)}(\zeta_i) + \theta_j * F_j^{(r,\ell)}(\zeta_i)\right)$$

with non-negligible probability. Thus, by the Schwartz-Zippel lemma, for all $\ell \in [m]$, $\mathsf{mac}_{j,i}^{(r,\ell)} = \mathsf{mac}_{j,i}^{(s,\ell)}$, which completes the proof.

4.3 Efficient Simultaneous Openings a la [26] and [29]

To get an improved communication complexity, a first idea is to use the robust decoding idea of [26] which works for fewer than $n/3$ malicious parties (and for fewer than $n/3$ simultaneous openings). To implement this idea for fewer than $n/2$ malicious parties (for a degree $n/2$ Shamir sharing) using the precomputed authentication information requires non-trivial advances, as demonstrated in [29] (and requires $\Omega(n^2)$ simultaneous openings). Since our robust secret-sharing scheme is different from [29] and has less correlated structure (in order to get a more efficient preprocessing), the simultaneous opening requires further new ideas. Remarkably, our robust secret-sharing scheme not only allows such amortized communication-complexity gain, it is only a four round protocol as opposed to the seven round protocol of [29]. Further, while [29] requires total communication $4n^3$ $\mathbb{E}$-elements (for $n^2/2$ simultaneous openings), our protocol requires $2n^3$ $\mathbb{F}$-elements and $2n^3$ $\mathbb{E}$-elements (for $n^2/2$ simultaneous openings).

The protocol $\Pi_{\mathsf{RSS}}.\mathtt{OpenSoundSimul}$, described in Supplementary Material B in detail, takes input that consists of parties having $\{[\![a^{(m,k)}]\!]\}_{m\in[n],k\in[t+1]}$. Since our main result is a linear communication complexity robust opening for only $\Omega(n)$ simultaneous openings, described next in Section 5, we only state the robustness theorem here for this weaker setting, and the proof is deferred to Supplementary Material B.

Theorem 4.5. [Robustness] *Given $n(t+1)$ robust sharings $\{[\![a^{(m,k)}]\!]\}_{m\in[n],k\in[t+1]}$, at the end of the above decoding procedure, each honest party has $\{a^{(m,k)}\}_{m\in[n],k\in[t+1]}$, except with probability upper bounded by $nt(t+2)/(|\mathbb{E}|-1)$.*

One can consider the same robust secret-sharing scheme, but where the degree t of the polynomials $F_0(X), F_1(X), \ldots, F_n(X)$ is set to $\lceil n/3\rceil - 1$. We will use $[\![\cdot]\!]^{n,n/3}$ to denote this instantiation and will make use of it during the input phase of our enhanced-correctness protocol. It follows from Lemma 4.3 that robustness is maintained against $\lceil n/2\rceil - 1$ colluding parties.

5 $\Theta(n)$ Simultaneous Robust Openings Using Three-Folded List Decoding

In this section, we give an efficient robust opening protocol that simultaneously opens $t+1$ sharings. The main idea will be to modify the OpenRobust procedure of [26] of Section 3.2. Instead of expanding t shares into n using a $n \times t$ Vandermonde matrix, we will expand into $3n$ using a $3n \times t$ Vandermonde matrix. This can be viewed as a three-folded code that groups three standard Reed-Solomon evaluations into a block that is associated with each party. In step 2, the parties will use the authenticators of our robust secret-sharing scheme to decode. Then, in step 3, they will list decode instead of sending authenticators, with the list being of size two. We use Guruswami–Sudan decoding [41] on the unfolded word as the general-purpose polynomial-time decoder. Finally, by choosing a challenge point on which these two openings disagree, the parties can efficiently obtain authenticated sharings of the correct opening at the challenge point. Now, we first define the three-folded code that will be used in our opening protocol and prove the size two list decoding bound.

5.1 The Three-Folded Code and List-Decoding Bound

Let $n = 2t+1$, let $1 \le W \le t+1$, and choose public, distinct points

$$\{\beta_{j,u} : j \in [n],\ u \in [3]\} \subseteq \mathbb{F}.$$

A polynomial $f \in \mathbb{F}[X]$ of degree less than W is encoded by its evaluations at these $3n$ points. Party P_j owns the block $\big(f(\beta_{j,1}), f(\beta_{j,2}), f(\beta_{j,3})\big)$.

For an arbitrary received block word $\boldsymbol{y} = (y_{j,u})_{j\in[n],u\in[3]} \in \mathbb{F}^{3n}$, define

$$\mathcal{L}(\boldsymbol{y}) := \{g \in \mathbb{F}[X] : \deg(g) < W, \quad |\{j : g(\beta_{j,u}) = y_{j,u} \text{ for every } u \in [3]\}| \ge t+1\}. \tag{2}$$

Lemma 5.1 (List Size Bound). *For every received word $\boldsymbol{y}$, the list $\mathcal{L}(\boldsymbol{y})$ has size at most two.*

Proof. Suppose, toward a contradiction, that $\mathcal{L}(\boldsymbol{y})$ contains three distinct polynomials f_1, f_2, f_3. For $a \in \{1,2,3\}$, let

$$S_a := \{j \in [n] : f_a(\beta_{j,u}) = y_{j,u} \text{ for every } u \in [3]\}.$$

Then $|S_a| \ge t+1$ for every a. If $a \ne b$, each party in $S_a \cap S_b$ supplies three distinct roots of the nonzero polynomial $f_a - f_b$. Since $\deg(f_a - f_b) \le W-1 \le t$, $|S_a \cap S_b| \le \left\lfloor \frac{W-1}{3} \right\rfloor \le \left\lfloor \frac{t}{3} \right\rfloor$.

Consequently,

$$\sum_{1\le a<b\le 3} |S_a \cap S_b| \le 3\left\lfloor \frac{t}{3} \right\rfloor \le t. \tag{3}$$

For each party j, put $m_j := |\{a \in \{1,2,3\} : j \in S_a\}|$. Double counting and $\binom{m}{2} \ge m-1$ for $m \in \{0,1,2,3\}$ give $\sum_{1\le a<b\le 3} |S_a \cap S_b| = \sum_{j\in[n]} \binom{m_j}{2}$, which is at

least $\sum_{j\in[n]}(m_j-1) = \sum_{a=1}^{3}|S_a| - n$, which in turn is seen to be at least $t+2$, contradicting Eq. (3). Hence the list has size at most two. Observe that the argument applies to every fixed received word, including one in which malicious parties choose their blocks after seeing all honest blocks.

The proof above is Johnson-type in that it double-counts pairwise agreements among codewords lying in the same Hamming ball. It is not, however, simply the usual Johnson bound: standard formulations use a Cauchy–Schwarz relaxation to handle arbitrary list sizes, whereas the exact inequality $\binom{m_i}{2} \geq m_i - 1$ is tailored to ruling out three candidates and gives the sharper finite-parameter conclusion that the list has size at most two. Thus the lemma may be viewed as an elementary list-size-two Johnson-type bound for the 3-folded code.

Lemma 5.2 (Efficient List Decoding). *For every received word $\boldsymbol{y}$, the list $\mathcal{L}(\boldsymbol{y})$ of size two is efficiently computable.*

The proof is deferred to Supplementary Material C.

5.2 Protocol for Efficient Simultaneous Robust Opening of $t+1$ Sharings

We now give an efficient robust opening protocol that simultaneously opens $t+1$ sharings. Let $\mathbf{G}(X,Y) = \sum_{k\in[t+1]} \mathbf{F}^{(k)}(X)\cdot Y^{k-1}$. Let $\{\beta_{j,u}\}_{j\in[n],\ u\in[3]}$ be $3n$ fixed (and public) distinct elements of $\mathbb{F}$. The set of points $\{\beta_{j,u}\}_{u\in[3]}$ are assigned to P_j for the 3-folded code. WLOG, we can have $\beta_{j,1} = \zeta_j$. Note that this requires $|\mathbb{F}| \geq 3n$. The list decoder will take advantage of the fact that all 3 elements sent by honest parties will be correct. For a received codeword $\boldsymbol{c} = (c_{j,u})_{j\in[n],u\in[3]}$, define $\mathcal{L}(\boldsymbol{c})$ as in Eq. 2, where $W = t+1$.

We next compress the authentication of the three symbols reconstructed by one owner. Fix $i \neq j$. For $u \in [3]$, let $X_{i,j,u} = G_0(\zeta_i, \beta_{j,u})$, $Y_{i,j,u} = G_j(\zeta_i, \beta_{j,u})$, and $M_{i,j,u} = \sum_{k=1}^{t+1} \beta_{j,u}^{k-1} \mathsf{mac}_{j,i}^{(k)} = X_{i,j,u} + \theta_j Y_{i,j,u}$.

Party P_j computes every $M_{i,j,u}$ locally from its existing tags. Define

$$\Phi_3(y_1, y_2, y_3) \coloneqq y_1 + y_2^2 + y_3^3. \tag{4}$$

Instead of sending all the X and Y values, an honest P_i sends

$$X_{i,j,1}, X_{i,j,2}, X_{i,j,3}, \qquad S_{i,j} \coloneqq \Phi_3(Y_{i,j,1}, Y_{i,j,2}, Y_{i,j,3}). \tag{5}$$

On receiving a purported packet $(\widehat{X}_1, \widehat{X}_2, \widehat{X}_3, \widehat{S})$, P_j computes, for $u \in [3]$,

$$\widehat{Y}_u \coloneqq \frac{M_{i,j,u} - \widehat{X}_u}{\theta_j} \tag{6}$$

and accepts the packet precisely when $\widehat{S} = \Phi_3(\widehat{Y}_1, \widehat{Y}_2, \widehat{Y}_3)$.

Robust Opening Protocol
(for $t+1$ Simultaneous Sharings)

Input. The parties have $\{[\![a^{(k)}]\!]\}_{k\in[t+1]}$.

Round 1: For every ordered pair $i \neq j$, party P_i sends P_j the compressed packet (5). Party P_j includes its own three locally computed X values as valid evaluations and applies the compressed check following (6) to each received packet as a whole. If fewer than $t+1$ senders are accepted, it outputs failure. Otherwise, for each $u \in [3]$, it interpolates the degree-t polynomial $G_0(X, \beta_{j,u})$ from the X values of any $t+1$ accepted senders, and records

$$c_{j,u} := G_0(0, \beta_{j,u}).$$

Round 2: Each P_i sends its three-symbol block $(c_{i,1}, c_{i,2}, c_{i,3})$ to every other party P_j. Let $\widehat{\mathbf{c}}_j$ denote the codeword received by P_j by concatenating all received three-symbol blocks and its own local block. P_j computes

$$\mathcal{L}_j := \mathcal{L}(\widehat{\mathbf{c}}_j).$$

Missing or malformed blocks are replaced by an arbitrary fixed block.

Round 3: Each P_i chooses $\rho_i \in \mathbb{F}$ as follows. If $\mathcal{L}_i = \{g_0, g_1\}$ has two members, it chooses one of the $3n$ $\beta_{j,u}$'s for which $g_0(\rho_i) \neq g_1(\rho_i)$. Such a point exists since the nonzero polynomial $g_0 - g_1$ has degree at most $t < 3n$ and therefore cannot vanish at all $3n$ outer points. If $|\mathcal{L}_i| \leq 1$, it sets $\rho_i := \beta_{1,1}$. P_i sends ρ_i to every other party.

Round 4: On receiving a challenge $\rho_j \in \mathbb{F}$ from P_j, party P_i sends

$$A_{i,j} := G_0(\zeta_i, \rho_j), \qquad B_{i,j} := G_j(\zeta_i, \rho_j) \tag{7}$$

to P_j. Party P_j includes its own locally computed pair as valid and accepts a received pair precisely when

$$A_{i,j} + \theta_j B_{i,j} = \sum_{k=1}^{t+1} \rho_j^{k-1} \mathsf{mac}_{j,i}^{(k)}. \tag{8}$$

If fewer than $t+1$ evaluations are accepted, it outputs failure. Otherwise, it interpolates the degree-t polynomial $G_0(X, \rho_j)$ from any $t+1$ accepted evaluations and obtains

$$z_j := G_0(0, \rho_j). \tag{9}$$

It outputs the coefficient of Y^{k-1} as $a^{(k)}$ for every $k \in [t+1]$, where the coefficients are taken from the unique $g \in \mathcal{L}_j$ satisfying $g(\rho_j) = z_j$. If there is not exactly one such polynomial, it outputs failure.

Note that the synchronous barrier before Round 3 is essential. It ensures that the block word of each honest receiver, including all reports of rushing corrupt

owners, is fixed before that receiver chooses a point separating the candidates in that list.

Correctness and communication. We first show compressing the Y values into a single sent value in round 1 still gives soundness.

Lemma 5.3 (Soundness of Compressed Authentication). *Suppose P_j is honest and θ_j is uniform in $\mathbb{E} \setminus \{0\}$ conditioned on the adversary's view. If $(\widehat{X}_1, \widehat{X}_2, \widehat{X}_3) \neq (X_{i,j,1}, X_{i,j,2}, X_{i,j,3})$, then the packet is accepted with probability at most $3/(|\mathbb{E}| - 1)$.*

The proof is deferred to Supplementary Material C.

We now prove the correctness and communication cost of the opening protocol.

Theorem 5.4. *Let $t = \lceil n/2 \rceil - 1$, and let at most t parties be statically corrupted. The protocol above robustly opens $\{[\![a^{(k)}]\!]\}_{k \in [t+1]}$ in four rounds with failure probability at most $\frac{4t(n-t)}{|\mathbb{E}|-1}$. The total communication is at most*

$$8n(n-1) \quad \mathbb{F}\text{-elements} \quad + \quad 2n(n-1) \quad \mathbb{E}\text{-elements}.$$

The proof is deferred to Supplementary Material C.

6 Ideal Functionality for Extra-Correct Secure-Function-Evaluation (X-SFE)

In this section, we define an ideal functionality for secure function evaluation (SFE) in the multi-client distributed-server setting. In particular, we consider boosting the correctness threshold to $n/2$ for a functionality which usually provides security against $n/3$ malicious parties. The ideal functionality $\mathcal{F}_{\mathsf{X\text{-}SFE}[\mathbb{F},f,n,m]}$ is defined in Supplementary Material D in Figure 3.

7 Input Sharing

In this section, we describe how clients can share their inputs amongst the servers in our MPC protocols. At a high level, this is done by first having the servers run a preprocessing step where each server S_i receives a share $[\![\rho]\!]_i$ for a random $\rho \in \mathbb{F}$ according to the robust secret-sharing scheme of Figure 1. Once this preprocessing is finished, each server S_i sends $[\![\rho]\!]_i$ to the client P, from which P can emulate $\Pi_{\mathsf{RSS}}.\mathtt{OpenSound}$ for each server to obtain $\rho^{(i)}$. If a majority of the $\rho^{(i)}$'s are equal to some ρ, then P will Byzantine broadcast $x - \rho$ to the servers, where x is its input. From this, each server can obtain $[x]_i$ with respect to the underlying Shamir secret-sharing scheme. For our $n/3$-private protocol with enhanced correctness, we use $[\![\cdot]\!]^{n,n/3}$ as our instantiation of the robust secret-sharing scheme.

The detailed ideal functionality for input sharing $\mathcal{F}_{\mathsf{inputShare}}$, as well as its realization in the preprocessing model, is deferred to Supplementary Material E.

8 Our Honest-Majority Fully-Secure MPC Protocol

In this section, we describe our honest-majority protocols (Thms. 1.1 and 1.2). These protocols will use our robust secret-sharing scheme and preprocessing material consisting of Beaver multiplication tuples. We now describe the preprocessing ideal functionality, which we call $\mathcal{F}_{\mathsf{Beaver}}$. For Beaver multiplication tuples [7], the ideal functionality generates fresh random $\langle\!\langle l_k \rangle\!\rangle$, $\langle\!\langle r_k \rangle\!\rangle$, $\langle\!\langle l_k r_k \rangle\!\rangle$, all of these using the same evaluation points $\vec{\theta}$. Formal description of $\mathcal{F}_{\mathsf{Beaver}}$ is in Fig. 5 in Supplementary Material F. This functionality can be implemented by any of the known MPC protocols that are secure against up to $n/2$ malicious corruptions, such as [39, 22, 4]. We give such an instantiation in Section 10.

The online phase of the protocol follows the standard use of Beaver tuples to evaluate multiplication gates (see e.g. [26, Fig. 6]). The proof of security of this protocol is standard [6] assuming the secure realization of $\mathcal{F}_{\mathsf{Beaver}}$. Note that the circuit width must be $\Omega(n^2)$ in order to use the simultaneous opening procedure described in Section 4 and $\Omega(n)$ to use the one described in Section 5.

9 A Novel $n/3$-Private and Robust MPC Protocol with Honest-Majority Correctness

In this section, we show Theorem 1.3 by describing a protocol that UC-realizes the extra-correct secure-function evaluation functionality $\mathcal{F}_{\mathsf{X\text{-}SFE}}$ of Section 6. Since this protocol will use global-authenticator a la SPDZ, where a global secret α is used to pre-authenticate various Beaver tuples, the input sharing must also use the same global authenticator α. In other words, the input sharing scheme, such as the one that realizes $\mathcal{F}_{\mathsf{inputShare}}[\mathbb{F}, n, \langle t_p, t_r, t_c \rangle, [\,\cdot\,]]$ and gives shares to servers using the $[\,\cdot\,]$ parameterized sharing-scheme, must employ a sharing scheme that uses α-authenticated Beaver tuples. We now describe the preprocessing ideal functionality, which we call $\mathcal{F}_{\mathsf{X\text{-}Beaver}}$ (see Supplementary Materials G for detailed description). As is usual in offline-online MPC protocols, this preprocessing functionality just generates various Beaver multiplication tuples, as well as authenticated versions of these using the global MAC key α. As previously mentioned, it also generates sharings to be used by the input phase by calling $\mathcal{F}_{\mathsf{RSS\text{-}PRE}}$. This functionality can be implemented by any of the known MPC protocols that are secure against up to $n/2$ malicious corruptions, such as [39, 22, 4]. We give such an instantiation in Section 10.

Having described the preprocessing ideal functionality $\mathcal{F}_{\mathsf{X\text{-}Beaver}}$, we now describe the online protocol in the $\mathcal{F}_{\mathsf{X\text{-}Beaver}}$-hybrid model. It has a similar structure to that of the BGW online protocol using Beaver tuples, but it extends these tuples with authenticated tuples, which are used to later check in an attestation phase, a la SPDZ, whether more than $n/3$ (but $< n/2$) malicious servers caused an incorrect circuit evaluation. The scheme achieves an $n/3$ robustness threshold as it employs an underlying degree $n/3 - 1$ Shamir secret-sharing. The authenticated tags and global MAC key α are shared using the robust secret-sharing

scheme of Fig. 1 which assures decoding with even up to $< n/2$ corruptions (with overwhelming probability). The scheme is described next.

Protocol $\Pi_{\mathsf{ONLINE\text{-}X\text{-}SFE}[\mathbb{F},f,n,m]}$

Setup. Let the function f be represented by an arithmetic circuit C over $\mathbb{F}$, with a maximum of m inputs. A global coordinator GC invokes $\mathcal{F}_{\mathsf{X\text{-}Beaver}}$ $[\mathbb{F}, \mathbb{E}, n, m, |C|+1]$ to populate the n servers with

- Secrets $\theta_i \in \mathbb{E}$
- Input-converting authenticated tuples (one for each input wire)
- Extended and authenticated Beaver tuples (one for each multiplication gate in C and the output wire)
- $\langle\!\langle \alpha \rangle\!\rangle^{\mathbb{E}}$, $\langle\!\langle x \rangle\!\rangle^{\mathbb{E}}$, $\langle\!\langle \vec{\beta} \rangle\!\rangle^{\mathbb{E}}$

A maximum of m clients $P_1, P_2, ..., P_m$ share their inputs amongst the n servers using $\Pi_{\mathsf{inputShare}}$ $[\mathbb{F}, n, \langle n/3, n/2, n/2\rangle, [\cdot]]$ (which itself uses input-converting authenticated tuples generated by $\mathcal{F}_{\mathsf{X\text{-}Beaver}}$ $[\mathbb{F}, \mathbb{E}, n, m, |C|+1]$ above replacing the stand-alone invocation of $\mathcal{F}_{\mathsf{RSS\text{-}PRE}}$). Let $\ell \leq m$ be the number of clients that succeed in sharing their inputs. Observe that if client P_i has input a_i, then during the execution of $\Pi_{\mathsf{inputShare}}$ $[\mathbb{F}, n, \langle n/3, n/2, n/2\rangle, [\cdot]]$, the servers use the $[\![\rho_i]\!]$ portion of the input-converting authenticated tuple $([\![\rho_i]\!], \langle\!\langle \rho_i \cdot \alpha \rangle\!\rangle^{\mathbb{E}})$ and obtain $[a_i]$. Moreover, all servers also obtain the value $\hat{a}_i = a_i - \rho_i$ that is Byzantine broadcast by P_i. Letting $\rho'_i = -\rho_i$ and noting that $\langle\!\langle \rho'_i \cdot \alpha \rangle\!\rangle^{\mathbb{E}} = -\langle\!\langle \rho_i \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$, it follows that all servers have $\hat{a}_i = a_i + \rho'_i$ and their shares of $[a_i]$ and $\langle\!\langle \rho'_i \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$.

Online Stage. The servers proceed in $|C|$ stages for the $|C|$ multiplication gates. [More efficiently, the servers can handle all gates at the same depth simultaneously as that saves on communication when using `OpenRobust`.]

The computation will be described inductively. As the base case, which are the input wires, for each input a_i, the servers have the value $\hat{a}_i = a_i + \rho_i$ and shares of $[a_i], \langle\!\langle \rho_i \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$.

Inductively, assume that the servers have the value $\hat{a}_i = a_i + \rho_i$ and shares of $[a_i], \langle\!\langle \rho_i \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$ for an intermediate set of wires $\mathbb{I}$ that feed into an addition gate g. Addition is performed locally as follows:

Procedure $\mathsf{Add}(\{[a_i], \hat{a}_i, \langle\!\langle \rho_i \cdot \alpha \rangle\!\rangle^{\mathbb{E}}\}_{i \in \mathbb{I}})$:

1. Compute $[a_g] = \sum_{i \in \mathbb{I}} [a_i]$, $\hat{a}_g = \sum_{i \in \mathbb{I}} \hat{a}_i$, $\langle\!\langle \rho_g \cdot \alpha \rangle\!\rangle^{\mathbb{E}} = \sum_{i \in \mathbb{I}} \langle\!\langle \rho_i \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$
2. Return $[a_g], \hat{a}_g, \langle\!\langle \rho_g \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$.

Assume that a specific multiplication gate $g \in [\ell + 1..\ell + |C|]$ has one left input a_l, and one right input a_r, and outputs $a_g = a_l \cdot a_r$. Assume that the servers have the values $\hat{a_l} = a_l + \rho_l, \hat{a_r} = a_r + \rho_r$, and the shares

$[a_l], \langle\!\langle \rho_l \cdot \alpha \rangle\!\rangle^{\mathbb{E}}, [a_r], \langle\!\langle \rho_r \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$. They access an unused extended and authenticated Beaver tuple and compute $[a_g]$ as follows:
Procedure $\mathsf{Mult}([a_l], \hat{a}_l, \langle\!\langle \rho_l \cdot \alpha \rangle\!\rangle^{\mathbb{E}}, [a_r], \hat{a}_r, \langle\!\langle \rho_r \cdot \alpha \rangle\!\rangle^{\mathbb{E}})$:

1. Obtain a fresh extended and authenticated Beaver tuple, say
$$[l], [r], [lr], \langle\!\langle l \cdot \alpha \rangle\!\rangle^{\mathbb{E}}, \langle\!\langle r \cdot \alpha \rangle\!\rangle^{\mathbb{E}}, \langle\!\langle lr \cdot \alpha \rangle\!\rangle^{\mathbb{E}}.$$
2. Compute $[a_l + l], [a_r + r]$.
3. Open $a_l + l, a_r + r$ using $\mathsf{OpenRobust}$.
4. Compute $[a_g] = (a_l + l)(a_r + r) - (a_l + l)[r] - (a_r + r)[l] + [lr]$.
5. Compute $\langle\!\langle \rho_g \cdot \alpha \rangle\!\rangle^{\mathbb{E}} = (a_l + l)\langle\!\langle r \cdot \alpha \rangle\!\rangle^{\mathbb{E}} + (a_r + r)\langle\!\langle l \cdot \alpha \rangle\!\rangle^{\mathbb{E}} - \langle\!\langle lr \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$.
6. Compute $\langle\!\langle P_g \rangle\!\rangle^{\mathbb{E}} = \langle\!\langle \rho_l \cdot \alpha \rangle\!\rangle^{\mathbb{E}} - \langle\!\langle l \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$, $\langle\!\langle Q_g \rangle\!\rangle^{\mathbb{E}} = \langle\!\langle \rho_r \cdot \alpha \rangle\!\rangle^{\mathbb{E}} - \langle\!\langle r \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$.
7. Compute $\hat{a}_g = (a_l + l)(a_r + r)$.
8. Compute $R_g = \hat{a}_l - (a_l + l)$, and $S_g = \hat{a}_r - (a_r + r)$.
9. Return $[a_g], \hat{a}_g, \langle\!\langle \rho_g \cdot \alpha \rangle\!\rangle^{\mathbb{E}}, \langle\!\langle P_g \rangle\!\rangle^{\mathbb{E}}, R_g, \langle\!\langle Q_g \rangle\!\rangle^{\mathbb{E}}, S_g$

(*Remark:* While $[a_g]$, $\hat{a}_g$, $\langle\!\langle \rho_g \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$ are used iteratively, the rest are saved for the attestation phase.)

Output Wire. For the output wire w, open $[a_w + l]$, using an authenticated Beaver tuple but ignore $[r]$ etc. Similarly, compute R_w and ignore computing S_w, as well as compute $\langle\!\langle P_w \rangle\!\rangle^{\mathbb{E}}$ and ignore $\langle\!\langle Q_w \rangle\!\rangle^{\mathbb{E}}$. In the attestation below, treat Q_w and S_w for this output wire as zero.

Attestation Phase.

i. The n servers open $\langle\!\langle x \rangle\!\rangle^{\mathbb{E}}$ with $\Pi_{\mathsf{RSS}}.\mathtt{OpenSound}$ (of Section 4).
ii. Now the servers locally compute:
$$\langle\!\langle A \rangle\!\rangle^{\mathbb{E}} = \sum_{g=\ell+1}^{\ell+|C|+1} x^{2(g-\ell)} \langle\!\langle P_g \rangle\!\rangle^{\mathbb{E}} + x^{2(g-\ell)+1} \langle\!\langle Q_g \rangle\!\rangle^{\mathbb{E}},$$
$$B = \sum_{g=\ell+1}^{\ell+|C|+1} x^{2(g-\ell)} R_g + x^{2(g-\ell)+1} S_g.$$
iii. Recall, each server S_i's share of $\langle\!\langle A \rangle\!\rangle^{\mathbb{E}}$ contains $F_0(\zeta_i)$ for a polynomial $F_0(X)$ whose constant term is A. Let $v_i = F_0(\zeta_i) + \beta_i$ where $\beta_i \in \mathbb{E}$ is part of $\langle\!\langle \vec{\beta} \rangle\!\rangle^{\mathbb{E}}$ generated in step 5 of $\mathcal{F}_{\mathsf{X\text{-}Beaver}}$. Server S_i also has $F_j(\zeta_i)$. Server S_i sends $(v_i, F_j(\zeta_i))$ to server S_j. This step is same as the Open-Start part of $\Pi_{\mathsf{RSS}}.\mathtt{OpenSound}$, except that $F_0(\zeta_i)$ has been replaced by v_i.
iv. The n servers open $\langle\!\langle \vec{\beta} \rangle\!\rangle^{\mathbb{E}}$ with $\Pi_{\mathsf{RSS}}.\mathtt{OpenSound}$. Each server S_i either gets β_j or $\perp$ (for all $j \in [n]$). For each server S_i, let J_i be the set of servers $j \in [n]$ that had the β_j verified by $\Pi_{\mathsf{RSS}}.\mathtt{OpenSound}$. For all $j \in J_i$, each server S_i updates its received value $v^{(j)}$ (in the previous

step) by subtracting β_j from it, and saves it as $F_0(\zeta_j)$. For $j \notin J_i$, it sets message received from P_j as $\perp$. Each server now runs the verification and reconstruction part of $\Pi_{\mathsf{RSS}}.\mathtt{OpenSound}$ (for $\langle\!\langle A \rangle\!\rangle^{\mathbb{E}}$) to get A.

v. The n servers engage with the other servers to open α from $\langle\!\langle \alpha \rangle\!\rangle^{\mathbb{E}}$ using $\Pi_{\mathsf{RSS}}.\mathtt{OpenSound}$. If $\alpha = 0$, the server aborts.

vi. Each server checks whether $A \stackrel{?}{=} B \cdot \alpha$. If the check fails, set $\hat{a}_{\ell+|C|}$ to $\perp$.

vii. Servers engage with other servers to open $l \cdot \alpha$, where l is as in the output wire description, from $\langle\!\langle l \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$ using $\Pi_{\mathsf{RSS}}.\mathtt{OpenSound}$. Next, each server divides by α to get l and then computes $a_{\ell+|C|} = \hat{a}_{\ell+|C|} - l$ (if $\hat{a}_{\ell+|C|}$ was $\perp$ then $a_{\ell+|C|} = \perp$). If $a_{\ell+|C|}$ is not $\perp$, the server outputs this value to GC.

viii. Servers engage in an honest-majority Byzantine consensus on $a_{\ell+|C|}$, and output the consensus a or $\perp$ to GC (see comment below).

GC*:* Once GC (asynchronously) receives a value that has frequency at least $n/2$, it outputs this value to environment $\mathcal{E}$. If it receives enough responses so that it is assured that no one value will get frequency $n/2$, it outputs $\perp$ to the environment. Note that if no party misbehaved or erred, then in step (vii) itself more than $n/2$ parties would have output the same value to GC, and the environment would have got the output from GC even before the step (viii)'s Byzantine consensus was complete.

9.1 UC Security Proof

Theorem 9.1. *The protocol $\Pi_{\mathsf{ONLINE\text{-}X\text{-}SFE}[\mathbb{F},f,n,m]}$ UC-realizes ideal functionality $\mathcal{F}_{\mathsf{X\text{-}SFE}[\mathbb{F},f,n,m]}$ in the $\mathcal{F}_{\mathsf{X\text{-}Beaver}}$, $\mathcal{F}_{\mathsf{inputShare}}$, $\mathcal{F}_{\mathsf{BC}}$-hybrid model, where $\mathcal{F}_{\mathsf{BC}}$ denotes the ideal functionality for honest-majority Byzantine consensus.*

The simulator for the proof of this theorem as well as the full proof of the above theorem can be found in Supplementary Material H.

10 Preprocessing Phase

In this section, we give an instantiation of the preprocessing phases of our MPC protocols, $\mathcal{F}_{\mathsf{Beaver}}$ and $\mathcal{F}_{\mathsf{X\text{-}Beaver}}$, using [39]. The robust secret-sharing scheme used by the ideal functionality $\mathcal{F}_{\mathsf{Beaver}}$ has privacy by Theorem 4.2 and robustness by Lemma 4.3. We also give an instantiation of the preprocessing phase of [29] for comparative purposes.

10.1 Introduction to "Guaranteed Output Delivery Comes Free in Honest Majority MPC" [39]

The MPC protocol of [39] is an unconditionally secure honest-majority MPC protocol with guaranteed output delivery and $O(n|C|)$ total communication. As we will see, any such scheme with these properties will suffice. Let c_I, c_M, c_R,

and c_O be the number of input gates, multiplication gates, random gates, and output gates, respectively. We let $|C| = c_I + c_M + c_R + c_O$ be the size of the circuit. In our case, there are no private inputs (for our preprocessing phase), so $c_I = 0$. In [39], they only consider output gates that deliver output to all parties. In our application, we need another kind of output gate which opens the output to a single target party. However, this is easily handled as the output protocol in [39] anyway chooses one party P_{king} for accumulation, decoding, and dissemination, and so for our targeted delivery, we can just let P_{king} be that target party (without the dissemination part).

The MPC protocol in [39] achieves $(n/2, n/2, n/2)$-security with total communication complexity $O(n \cdot \phi \cdot |C| + n^3\lambda^2 + n^6 \cdot \lambda)$ bits plus $O(n^3 \cdot \lambda)$ bits on the assumed broadcast channel. Here $\phi = \log|\mathbb{F}|$ and λ is the security parameter as well as $\log|\mathbb{E}|$. Further, the proof of security of this dispute-control based protocol in [39] shows that if the protocol outputs all final outputs without failure, then *all internal gates were correctly computed in terms of sharings*. In other words, if the final output gates are output gates of a circuit C consisting of random gates, multiplication gates, and output gates, then if the protocol exits without failing, then the output of each random gate (in the rerun which caused no dispute) is indeed a t-sharing of a randomly distributed number in $\mathbb{F}$ (or $\mathbb{E}$ if a random number in $\mathbb{E}$ is sought). Similarly, the output of a multiplication gate is indeed a t-sharing of ab, where the inputs are (inductively) correct t-sharings of a and b. This inductive property of their proof is not commonly highlighted, but we will use it in our complexity analysis.

We remark that addition gates and multiplication by public scalars do not require any communication as these can be instantiated by local computation of each party. Using this result, we show the communication complexity of realizing our preprocessing ideal functionalities, $\mathcal{F}_{\mathsf{Beaver}}$ and $\mathcal{F}_{\mathsf{X\text{-}Beaver}}$.

10.2 Honest-Majority Protocols

In our honest-majority protocols, the preprocessing material consists of Beaver tuples $(\langle\!\langle l \rangle\!\rangle, \langle\!\langle r \rangle\!\rangle, \langle\!\langle lr \rangle\!\rangle)$, one for each multiplication gate in the circuit. Recall, double angle brackets are used for honest-majority robust shares, in particular the robust secret-sharing scheme of Fig 1 with $n = 2t+1$. Additionally, it contains additional shares of random values for the input phase and for the randomness needed during the simultaneous opening protocol for the robust secret-sharing scheme.

The honest-majority protocol of [29] is identical except it uses a different robust secret-sharing scheme (see Fig. 6 in Supplementary Material L). We note that a party P_j's share in the robust sharing scheme of Fig. 6 is $3(t+1)$ elements of $\mathbb{E}$ (or $2(t+1)$ elements per additional share, since the $(t+1)$ elements $\mathbf{A}(\zeta_j)$ can be reused). In constrast, under our robust sharing scheme of Fig. 1, a party P_j's share consists of $2n$ elements of $\mathbb{E}$ and 1 element of $\mathbb{F}$ (with 1 element of $\mathbb{E}$ reusable across shares). Observe that the preprocessing material in our $n/2$ protocol is approximately twice as large as that of [29]. However, as we will show, our preprocessing material can actually be generated with a factor of n

less communication, thus giving us a factor of n improvement in the complexity of the offline phase. See Supplementary Material L for an analysis of complexity of preprocessing-MPC for generation of robust secret-shares as per the scheme in [29]. Here, we will focus on analysis of the complexity of generation of robust secret-shares as per our robust secret-sharing scheme. We note that our robust secret-sharing scheme uses two fields. However, in [39], an random gate over $\mathbb{E}$ can easily be converted into a random gate over $\mathbb{F}$ by fixing a surjective $\mathbb{F}$-linear map $\pi : \mathbb{E} \to \mathbb{F}$ and applying it sharewise since the public evaluation points ζ_i's are in $\mathbb{F}$.

We give a subroutine Augment to convert a sharing $\langle s \rangle$ to $\langle\langle s \rangle\rangle$ using sharings of secret points $\langle \theta_1 \rangle, \ldots, \langle \theta_n \rangle$, where $\langle\langle \cdot \rangle\rangle$ refers to the robust secret-sharing scheme of Section 4. Note that in our honest-majority protocol $n = 2t + 1$, and we use Augment to refer to this subroutine at that threshold. However, the same procedure can also be used to convert Shamir secret-shares of degree $n/3$, i.e. ${}^{n/3}[s]$ to $[\![s]\!]^{n,n/3}$ which we will use in the preprocessing for our $n/3$-protocol with enhanced correctness.

Augment

Given: A sharing $\langle s \rangle$ using public evaluation points $\zeta_1, \ldots, \zeta_n$ and sharings of secret evaluation points $\langle \theta_1 \rangle, \ldots, \langle \theta_n \rangle$.
Output: An augmented sharing $\langle\langle s \rangle\rangle$ using evaluation points $\theta_1, \ldots, \theta_n$.
Protocol:
1. Let $\langle F_{00} \rangle = \langle s \rangle$.
2. Parties invoke $(n+1)(t+1) - 1$ random gates to get the remaining $\langle F_{ij} \rangle$, for $i \in \{0, \ldots, n\}, j \in \{0, \ldots, t\}$ with each F_{ij} random and independent in $\mathbb{E}$ for $i \neq 0$ and in $\mathbb{F}$ for $i = 0$. Define $F_i(X) = \sum_{j=0}^{t} F_{ij} X^j$.
3. Parties locally compute (using public additions and scalar multiplications) $\langle F_i(\zeta_j) \rangle$ for all $i \in \{0, \ldots, n\}$ and $j \in [n]$.
4. For all $i, j \in [n]$ with $i \neq j$, parties compute $\langle \mathsf{mac}_{j,i} \rangle$ using one multiplication gate (between $\langle \theta_j \rangle$ and $\langle F_j(\zeta_i) \rangle$) and a local addition.
5. Parties use output gates to output the shares to the appropriate parties (e.g., $\langle F_i(\zeta_j) \rangle$ is opened to P_j).

Observe that Augment requires $(n + 1)(t + 1) - 1$ random gates, $n(n - 1)$ multiplication gates, and $2n^2$ output gates. In total, it requires a circuit of size $O(n^2)$ to compute inside the MPC protocol. Overall, the preprocessing for our honest-majority protocol (Thm. 1.2) requires a circuit of size $O((m + \ell)n^2)$ to compute. Thus, the preprocessing material can be generated in communication $O(n^3|C|) + \mathsf{poly}(n)$ using [39]. Observe that we save a factor n communication compared to the preprocessing generation for [29]. The full analysis is deferred to Supplementary Material J. In Supplementary Material K we do a similar detailed analysis of preprocessing cost of generating extended and authenticated Beaver tuples for the enhanced-correctness-$n/3$ protocol. In particular, we analyze instantiating $\mathcal{F}_{\mathsf{X\text{-}Beaver}}$.

Statement of AI Usage. Most of the work in this paper is the original work of the authors. In particular, AI was not used for the construction of our robust secret-sharing scheme or Thms. 1.2 or 1.3. However, in an attempt to prove a $\omega(n^2)$ total communication lower bound for $\Theta(n)$ simultaneous openings using ChatGPT 5.6 Sol-Ultra in a robust linear secret sharing model formalized by the authors, AI eventually converged on a list decoding argument after a back and forth with the authors. We simplified the argument and wrote the protocol of Section 5 and in the process have thoroughly verified correctness.

References

1. Abraham, I., Asharov, G., Patil, S., Patra, A.: Detect, pack and batch: Perfectly-secure MPC with linear communication and constant expected time. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023. pp. 251–281. Springer Nature Switzerland, Cham (2023)
2. Abraham, I., Asharov, G., Yanai, A.: Efficient perfectly secure computation with optimal resilience. In: Nissim, K., Waters, B. (eds.) Theory of Cryptography. pp. 66–96. Springer International Publishing, Cham (2021)
3. Acharya, A., Hazay, C., Venkitasubramaniam, M.: On achieving "best-in-the-multiverse" MPC. In: Theory of Cryptography Conference. pp. 119–151. Springer (2025)
4. Agarwal, A., Bienstock, A., Damgård, I., Escudero, D.: Honest Majority GOD MPC with $O(\mathsf{depth}(C))$ Rounds and Low Online Communication. Cryptology ePrint Archive, Paper 2024/1479 (2024), https://eprint.iacr.org/2024/1479
5. Asharov, G., Lindell, Y.: A full proof of the BGW protocol for perfectly secure multiparty computation. J. Cryptol. **30**(1), 58–151 (Jan 2017), https://eprint.iacr.org/2011/136.pdf
6. Bar-Ilan, J., Beaver, D.: Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In: Rudnicki, P. (ed.) 8th ACM PODC. pp. 201–209. ACM (Aug 1989). https://doi.org/10.1145/72981.72995
7. Beaver, D.: Multiparty protocols tolerating half faulty processors. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 560–572. Springer, New York (Aug 1990). https://doi.org/10.1007/0-387-34805-0_49
8. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Advances in Cryptology—CRYPTO'91. pp. 420–432. Springer (1991)
9. Beaver, D.: Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. J. Cryptol. **4**(2), 75–122 (1991)
10. Beerliová-Trubíniová, Z., Hirt, M.: Efficient multi-party computation with dispute control. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 305–328. Springer, Berlin, Heidelberg (Mar 2006). https://doi.org/10.1007/11681878_16
11. Beerliová-Trubíniová, Z., Hirt, M.: Perfectly-secure MPC with linear communication complexity. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 213–230. Springer, Berlin, Heidelberg (Mar 2008). https://doi.org/10.1007/978-3-540-78524-8_13
12. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: 20th ACM STOC. pp. 1–10. ACM Press (May 1988). https://doi.org/10.1145/62212.62213

13. Ben-Sasson, E., Fehr, S., Ostrovsky, R.: Near-linear unconditionally-secure multiparty computation with a dishonest minority. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 663–680. Springer, Berlin, Heidelberg (Aug 2012). https://doi.org/10.1007/978-3-642-32009-5_39
14. Bishop, A., Pastro, V., Rajaraman, R., Wichs, D.: Essentially optimal robust secret sharing with maximal corruptions. In: Fischlin, M., Coron, J. (eds.) Advances in Cryptology - EUROCRYPT. Lecture Notes in Computer Science, vol. 9665, pp. 58–86. Springer (2016)
15. Boyle, E., Gilboa, N., Ishai, Y., Nof, A.: Efficient fully secure computation via distributed zero-knowledge proofs. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part III. LNCS, vol. 12493, pp. 244–276. Springer, Cham (Dec 2020). https://doi.org/10.1007/978-3-030-64840-4_9
16. Canetti, R.: Security and composition of multiparty cryptographic protocols. J. Cryptol. **13**(1), 143–202 (2000)
17. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. ECCC **TR01-016** (2001), https://eccc.weizmann.ac.il/eccc-reports/2001/TR01-016
18. Canetti, R., Rabin, T.: Universal composition with joint state. In: Boneh, D. (ed.) Advances in Cryptology - CRYPTO. Lecture Notes in Computer Science, vol. 2729, pp. 265–281. Springer (2003)
19. Carter, L., Wegman, M.N.: Universal classes of hash functions. J. Comput. Syst. Sci. **18**(2), 143–154 (1979)
20. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: 20th ACM STOC. pp. 11–19. ACM Press (May 1988). https://doi.org/10.1145/62212.62214
21. Chida, K., Genkin, D., Hamada, K., Ikarashi, D., Kikuchi, R., Lindell, Y., Nof, A.: Fast large-scale honest-majority MPC for malicious adversaries. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part III. LNCS, vol. 10993, pp. 34–64. Springer, Cham (Aug 2018). https://doi.org/10.1007/978-3-319-96878-0_2
22. Choudhury, A., Patra, A.: An efficient framework for unconditionally secure multiparty computation. IEEE Trans. Inf. Theor. **63**(1), 428–468 (Jan 2017)
23. Cramer, R., Damgård, I., Dziembowski, S., Hirt, M., Rabin, T.: Efficient multiparty computations secure against an adaptive adversary. In: Stern, J. (ed.) EUROCRYPT'99. LNCS, vol. 1592, pp. 311–326. Springer, Berlin, Heidelberg (May 1999). https://doi.org/10.1007/3-540-48910-X_22
24. Cramer, R., Damgård, I., Fehr, S.: On the cost of reconstructing a secret, or VSS with optimal reconstruction phase. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 503–523. Springer, Berlin, Heidelberg (Aug 2001). https://doi.org/10.1007/3-540-44647-8_30
25. Damgård, I., Ishai, Y., Krøigaard, M.: Perfectly secure multiparty computation and the computational overhead of cryptography. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 445–465. Springer, Berlin, Heidelberg (May / Jun 2010). https://doi.org/10.1007/978-3-642-13190-5_23
26. Damgård, I., Nielsen, J.B.: Scalable and unconditionally secure multiparty computation. In: Menezes, A. (ed.) Advances in Cryptology - CRYPTO 2007. Lecture Notes in Computer Science, vol. 4622, pp. 572–590. Springer (2007)
27. Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Annual Cryptology Conference. pp. 643–662. Springer (2012)

28. Dowsley, R., Müller-Quade, J., Otsuka, A., Hanaoka, G., Imai, H., Nascimento, A.C.A.: Universally composable and statistically secure verifiable secret sharing scheme based on pre-distributed data. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **94-A**(2), 725–734 (2011), https://eprint.iacr.org/2009/273.pdf
29. Escudero, D., Fehr, S.: On fully-secure honest majority MPC without n^2 round overhead. In: Aly, A., Tibouchi, M. (eds.) LATINCRYPT 2023. LNCS, vol. 14168, pp. 47–66. Springer, Cham (Oct 2023). https://doi.org/10.1007/978-3-031-44469-2_3
30. Fehr, S., Yuan, C.: Towards optimal robust secret sharing with security against a rushing adversary. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology - EUROCRYPT 2019. Lecture Notes in Computer Science, vol. 11478, pp. 472–499. Springer (2019)
31. Feldman, P., Micali, S.: An optimal probabilistic protocol for synchronous byzantine agreement. SIAM J. Comput. **26**(4), 873–933 (1997)
32. Furukawa, J., Lindell, Y., Nof, A., Weinstein, O.: High-throughput secure three-party computation for malicious adversaries and an honest majority. In: Coron, J.S., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part II. LNCS, vol. 10211, pp. 225–255. Springer, Cham (Apr / May 2017). https://doi.org/10.1007/978-3-319-56614-6_8
33. Gemmell, P., Naor, M.: Codes for interactive authentication. In: Stinson, D.R. (ed.) Advances in Cryptology - CRYPTO '93. Lecture Notes in Computer Science, vol. 773, pp. 355–367. Springer (1993)
34. Gemmell, P., Sudan, M.: Highly resilient correctors for polynomials. Inf. Process. Lett. **43**(4), 169–174 (1992)
35. Genkin, D., Ishai, Y., Prabhakaran, M., Sahai, A., Tromer, E.: Circuits resilient to additive attacks with applications to secure computation. In: Shmoys, D.B. (ed.) 46th ACM STOC. pp. 495–504. ACM Press (May / Jun 2014). https://doi.org/10.1145/2591796.2591861
36. Gennaro, R., Rabin, M.O., Rabin, T.: Simplified vss and fast-track multiparty computations with applications to threshold cryptography. In: Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing. PODC '98 (1998)
37. Goldwasser, S., Levin, L.A.: Fair computation of general functions in presence of immoral majority. In: Menezes, A., Vanstone, S.A. (eds.) Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings. Lecture Notes in Computer Science, vol. 537, pp. 77–93. Springer (1990)
38. Goyal, V., Liu, Y., Song, Y.: Communication-efficient unconditional MPC with guaranteed output delivery. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part II. LNCS, vol. 11693, pp. 85–114. Springer, Cham (Aug 2019). https://doi.org/10.1007/978-3-030-26951-7_4
39. Goyal, V., Song, Y., Zhu, C.: Guaranteed output delivery comes free in honest majority MPC. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 618–646. Springer, Cham (Aug 2020). https://doi.org/10.1007/978-3-030-56880-1_22
40. Guruswami, V., Patthak, A.C.: Correlated algebraic-geometric codes: Improved list decoding over bounded alphabets. Math. Comput. **77**(261), 447–473 (2008). https://doi.org/10.1090/S0025-5718-07-02012-1, https://doi.org/10.1090/S0025-5718-07-02012-1

41. Guruswami, V., Sudan, M.: List decoding algorithms for certain concatenated codes. In: 32nd ACM STOC. pp. 181–190. ACM Press (May 2000). https://doi.org/10.1145/335305.335327
42. Hirt, M., Maurer, U.M., Przydatek, B.: Efficient secure multi-party computation. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 143–161. Springer, Berlin, Heidelberg (Dec 2000). https://doi.org/10.1007/3-540-44448-3_12
43. Ishai, Y., Katz, J., Kushilevitz, E., Lindell, Y., Petrank, E.: On achieving the best of both worlds in secure multiparty computation. Cryptology ePrint Archive, Report 2010/029 (2010), https://eprint.iacr.org/2010/029
44. Ishai, Y., Kushilevitz, E., Prabhakaran, M., Sahai, A., Yu, C.H.: Secure protocol transformations. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part II. LNCS, vol. 9815, pp. 430–458. Springer, Berlin, Heidelberg (Aug 2016). https://doi.org/10.1007/978-3-662-53008-5_15
45. Katz, J., Koo, C.Y.: On expected constant-round protocols for byzantine agreement. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 445–462. Springer, Berlin, Heidelberg (Aug 2006). https://doi.org/10.1007/11818175_27
46. Lindell, Y., Nof, A.: A framework for constructing fast MPC over arithmetic circuits with malicious adversaries and an honest-majority. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 259–276. ACM Press (Oct / Nov 2017). https://doi.org/10.1145/3133956.3133999
47. Loss, J., Moran, T.: Combining asynchronous and synchronous byzantine agreement: The best of both worlds. Cryptology ePrint Archive (2018)
48. Pass, R., Shi, E.: Hybrid Consensus: Efficient Consensus in the Permissionless Model. In: Richa, A. (ed.) 31st International Symposium on Distributed Computing (DISC 2017). Leibniz International Proceedings in Informatics (LIPIcs), vol. 91, pp. 39:1–39:16 (2017)
49. Pass, R., Shi, E.: Thunderella: Blockchains with optimistic instant confirmation. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part II. LNCS, vol. 10821, pp. 3–33. Springer, Cham (Apr / May 2018). https://doi.org/10.1007/978-3-319-78375-8_1
50. Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In: 21st ACM STOC. pp. 73–85. ACM Press (May 1989). https://doi.org/10.1145/73007.73014
51. Shamir, A.: How to share a secret. Communications of the Association for Computing Machinery **22**(11), 612–613 (Nov 1979). https://doi.org/10.1145/359168.359176
52. Sudan, M.: Decoding of Reed–Solomon codes beyond the error-correction bound. J. Complex. **13**(1), 180–193 (1997)
53. Veldhuizen, B., Spini, G., Veugen, T., Kohl, L.: Extending the security of SPDZ with fairness. Proceedings on Privacy Enhancing Technologies **2024**(2), 330–350 (2024)

A Proof of Privacy [Theorem 4.2]

Proof. From the definition of $[\![a]\!]$, recall that $F_0(0) = a$. Assume, without loss of generality, that the adversary has shares of parties $P_1, ..., P_t$. To make the proof rigorous we introduce some notation. We use $\vec{F}^i$ as notation for (column) vector of coefficients of polynomial $F_i(X)$. Abusing earlier notation, we let $\mathbf{F}$ stand for the matrix consisting of columns $\vec{F}^0, ..., \vec{F}^n$. We let $\hat{\mathbf{F}}$ stand for the submatrix

with columns $\vec{F}^0, ..., \vec{F}^t$. We will use $\mathsf{diag}(\theta_1, ..., \theta_t)$ to stand for diagonal matrix with diagonal $\theta_1, ..., \theta_t$. We let $\mathbf{1}$ stand for all one column vector of length t. As before, $\mathbf{V}^{\{\zeta_1,...,\zeta_t\},p}$ stands for vandermonde matrix with degrees zero through $p-1$. Since the points ζ_i are clear for this lemma, we drop them from the superscript, and just write $\mathbf{V}^p$. The view of the adversary is as follows:

$$\begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_t \end{bmatrix} \quad \text{and} \quad \hat{\mathbf{F}} \cdot \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \theta_1 & 0 & \cdots & 0 \\ 0 & \theta_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & \theta_t \end{bmatrix} \quad \text{and} \quad \mathbf{V}^{t+1} \cdot \mathbf{F} \tag{10}$$

The first vector of elements is just the verification keys which are all chosen to be non-zero. The second set of elements come from the mac-tags that are provided to each party (see the definition of $\mathsf{mac}_{j,*}$ and how it can be computed; similar computational procedure gives the second set of elements). The third set of elements are the shares of $\mathbf{F}$ that the t parties have. Since $\vec{F}^{t+1}, ..., \vec{F}^n$ are independent of all other elements above as well as a, it suffices to show that the above view with $\mathbf{V}^{t+1} \cdot \mathbf{F}$ replaced by $\mathbf{V}^{t+1} \cdot \hat{\mathbf{F}}$ is independent of a.

For ease of exposition, we first break the $(t+1) \times (t+1)$ matrix $\hat{F}$ into four parts: (i) the top left element, which is just $\vec{F}_0^0 = a$, (ii) the remaining t entries of $\vec{F}^0$, which are denoted by $\vec{u}$, (iii) the top row of $\mathbf{F}$ except the left element a: these t elements will be denoted by $\vec{v}^\top$, and (iv) the remaining $t \times t$ matrix which we denote by $\mathbf{H}$. Thus,

$$\hat{\mathbf{F}} = \begin{bmatrix} \vec{F}_0^0 & \vec{v}^\top \\ \vec{u} & \mathbf{H} \end{bmatrix}$$

We next note that the second matrix above in (10) is same as $\vec{F}^0 \cdot \mathbf{1}^\top + [\vec{F}^1 \cdots \vec{F}^t] \cdot \mathsf{diag}(\theta_1, ..., \theta_t)$. Similarly, $\mathbf{V}^{t+1} \cdot \hat{\mathbf{F}}$ is same as $\mathbf{1} \cdot [\vec{F}_0^0 \;\; \vec{v}^\top] + \mathsf{diag}(\zeta_1, ..., \zeta_t) \cdot \mathbf{V}^t \cdot [\vec{u} \;\; \mathbf{H}]$, noting that the first column of $\mathbf{V}^{t+1}$ is just $\mathbf{1}$, and the remaining columns are same as $\mathsf{diag}(\zeta_1, ..., \zeta_t) \cdot \mathbf{V}^t$. This latter $t \times t$ matrix is thus invertible (since $\mathbf{V}^t$ is a square Vandermonde matrix with distinct evaluation points).

Looking at the rows of $\vec{F}^0 \cdot \mathbf{1}^\top + [\vec{F}^1 \cdots \vec{F}^t] \cdot \mathsf{diag}(\theta_1, ..., \theta_t)$, the top row is $\vec{F}_0^0 \cdot \mathbf{1}^\top + \vec{v}^\top \cdot \mathsf{diag}(\theta_1, ..., \theta_t)$. The rest of the rows can be written as $\vec{u} \cdot \mathbf{1}^\top + \mathbf{H} \cdot \mathsf{diag}(\theta_1, ..., \theta_t)$.

Similarly, looking at columns of $\mathbf{1} \cdot [\vec{F}_0^0 \;\; \vec{v}^\top] + \mathsf{diag}(\zeta_1, ..., \zeta_t) \cdot \mathbf{V}^t \cdot [\vec{u} \;\; \mathbf{H}]$, the first column is $\mathbf{1} \cdot \vec{F}_0^0 + \mathsf{diag}(\zeta_1, ..., \zeta_t) \cdot \mathbf{V}^t \cdot \vec{u}$. The rest of the columns are $\mathbf{1} \cdot \vec{v}^\top + \mathsf{diag}(\zeta_1, ..., \zeta_t) \cdot \mathbf{V}^t \cdot \mathbf{H}$. Thus, we are left with showing that the following are collectively independent of $\vec{F}_0^0$:

1. $\mathbf{T} \stackrel{\text{def}}{=} \mathsf{diag}(\theta_1, ..., \theta_t)$
2. $\vec{w}^\top \stackrel{\text{def}}{=} \vec{F}_0^0 \cdot \mathbf{1}^\top + \vec{v}^\top \cdot \mathsf{diag}(\theta_1, ..., \theta_t)$
3. $\vec{x} \stackrel{\text{def}}{=} \mathbf{1} \cdot \vec{F}_0^0 + \mathsf{diag}(\zeta_1, ..., \zeta_t) \cdot \mathbf{V}^t \cdot \vec{u}$
4. $\mathbf{Y} \stackrel{\text{def}}{=} \vec{u} \cdot \mathbf{1}^\top + \mathbf{H} \cdot \mathsf{diag}(\theta_1, ..., \theta_t)$
5. $\mathbf{Z} \stackrel{\text{def}}{=} \mathbf{1} \cdot \vec{v}^\top + \mathsf{diag}(\zeta_1, ..., \zeta_t) \cdot \mathbf{V}^t \cdot \mathbf{H}$

Now, note that $\mathbf{Z}$ can be obtained from $\mathbf{T}, \vec{w}^\top, \vec{x}, \mathbf{Y}$ using public quantities by the following identity (noting that $\mathbf{T}$ is invertible since all θ_i are non-zero)

$$\mathbf{Z} = \left(\mathsf{diag}(\zeta_1, ..., \zeta_t) \cdot \mathbf{V}^t \cdot \mathbf{Y} + \left(\mathbf{1} \cdot \vec{w}^\top - \vec{x} \cdot \mathbf{1}^\top\right)\right) \cdot \mathbf{T}^{-1}$$

So, we are left with showing that $\mathbf{T}, \vec{w}^\top, \vec{x}, \mathbf{Y}$ are independent of $\vec{F}_0^0$. Since $\mathbf{Y}$ has the additive term $\mathbf{H} \cdot \mathsf{diag}(\theta_1, ..., \theta_t)$, and the fully random and independent $t \times t$ matrix $\mathbf{H}$ does not occur anywhere else in expressions for $\mathbf{T}, \vec{w}^\top, \vec{x}$, we are left with showing independence of $\mathbf{T}, \vec{w}^\top, \vec{x}$ from $\vec{F}_0^0$. Each element of $\vec{w}^\top$ is independent of $\vec{F}_0^0$ by corollary 3.1, since $\vec{v}$ is independently random and all θ's are non-zero. Similarly, each element of $\vec{x}$ is independent of $\vec{F}_0^0$ as each element of $\mathsf{diag}(\zeta_1, ..., \zeta_t) \cdot \mathbf{V}^t \cdot \vec{u}$ is random and independent because $\vec{u}$ is random and independent. Finally, all the θ_i are chosen independent of $\vec{F}_0^0$, and that completes the proof.

B Protocol and Proof of Robust (Simultaneous) Opening [Theorem 4.5]

Protocol Π_{RSS}.OpenSoundSimul
(for Simultaneous Robust Opening of $n(t+1)$ Sharings)

Input: The parties have $\{[\![a^{(m,k)}]\!]\}_{m \in [n], k \in [t+1]}$. In the following, in the first two phases, i.e. Phase I and Phase II, all parties perform the protocol in parallel (and independently) for each $m \in [n]$. Hence, the superscript m will be suppressed for ease of exposition.

Phase I Redundant Encoding: Recall, each party P_i has shares $\mathbf{F}^{(k)}(\zeta_i)$ ($k \in [t+1]$). Party P_i treats these as a polynomial in Y (over $\mathbb{F} \times \mathbb{E}^n$. In fact, there is a better global view, where the $(t+1)$ many $\mathbf{F}^{(k)}(X)$ are treated as a bivariate polynomial $\mathbf{G}(X, Y) = \sum_{k \in [t+1]} \mathbf{F}^{(k)}(X) \cdot Y^{k-1}$. Then, it is easy to see that party P_i has $\mathbf{G}(\zeta_i, Y)$.
Party P_i expands its shares $\mathbf{G}(\zeta_i, Y)$ to a *vector* $\{\mathbf{G}(\zeta_i, \zeta_j)\}_{j \in [n]}$.

Phase I Sending: P_i sends only the 0-th and the j-th component of the j-th vector to P_j, i.e. it sends $\mathbf{G}_0(\zeta_i, \zeta_j)$, $\mathbf{G}_j(\zeta_i, \zeta_j)$ to P_j. To be perfectly clear, note that $\mathbf{G}_j(\zeta_i, \zeta_j) = \sum_{k \in [t+1]} F_j^{(k)}(\zeta_i) \cdot \zeta_j^{k-1}$.
Note: Party P_i communicates n $\mathbb{F}$-elements and n $\mathbb{E}$-elements in this phase (for each m).

Phase I Decoding: Each party P_j receives $\mathbf{G}_0(\zeta_i, \zeta_j)$ (for $i \in [n]$), and $\mathbf{G}_j(\zeta_i, \zeta_j)$ (for $i \in [n]$). Note each of these latter elements are a (public) linear combination of $F_j^{(k)}(\zeta_i)$ ($k \in [t+1]$, the combiners being ζ_j^{k-1}, and hence same for all i). Party P_j intends to decode and get the *same linear combination* of $\{F_0^{(k)}(0) = a^{(k)}\}_{k \in [t+1]}$, i.e. $\mathbf{G}_0(0, \zeta_j)$.

Recall, P_j has $\mathsf{mac}_{j,i}^{(k)}$ for each i (for each k). To validate party P_i's message, it tests (using its secret key θ_j):

$$\mathbf{G}_0(\zeta_i, \zeta_j) + \theta_j * \mathbf{G}_j(\zeta_i, \zeta_j) == \sum_{k=1}^{t+1} \mathsf{mac}_{j,i}^{(k)} * \zeta_j^{k-1}$$

Let I be the index set of parties P_i that pass the above test. If $|I| < t+1$ then abort. Otherwise, let S be any subset of I of size $t+1$. By applying the inverse of $\mathbf{V}^{S,t+1}$ to $\{\mathbf{G}_0(\zeta_i, \zeta_j)\}_{i \in S}$, P_j obtains the degree t polynomial $\mathbf{G}_0(X, \zeta_j)$, and hence also $\mathbf{G}_0(0, \zeta_j)$.
Note: Now, the parties collectively have evaluations of degree t polynomials $\mathbf{G}_0(0, Y)$ at points $\{\zeta_j\}_{j \in [n]}$.

Phase II Sending: Each party P_j sends to P_ℓ, its decoded $\mathbf{G}_0(0, \zeta_j)$.
Note: Each party sends n $\mathbb{F}$-elements (for each m).

Phase II Receiving: Each party P_ℓ now has $\{\mathbf{G}_0(0, \zeta_j)\}_j$.

In Phase III, we aggregate the material to be sent for the n different batches (i.e. for indices $m = 1..n$).

Phase III Common Randomness: The parties open a common random value ω (from pre-computed shares $[\![\omega]\!]$) using the simple-opening scheme $\Pi_{\mathsf{RSS}}.\mathtt{OpenSound}$ in Fig 2.

Phase III Sending: Each party P_i sends to P_ℓ,

$$\left\{\sum_{m=1}^{n} \omega^{m-1} * F_0^{(m,k)}(\zeta_i)\right\}_{k \in [t+1]} \text{ and } \left\{\sum_{m=1}^{n} \omega^{m-1} * F_\ell^{(m,k)}(\zeta_i)\right\}_{k \in [t+1]}.$$

Note: The total communication over all parties is $2 * (t+1) * n * n$ (indexing over k, i, ℓ) $\mathbb{E}$-elements. Since this is over $n * (t+1)$ openings, the amortized communication per opening per party is 2 elements of $\mathbb{E}$.

Phase III Receiving: Party P_ℓ overall receives $A_{i,k}$ and $B_{i,k}$ (with $i \in [n], k \in [t+1]$, which purportedly are

$$\left\{A_{i,k} = \sum_{m=1}^{n} \omega^{m-1} * F_0^{(m,k)}(\zeta_i)\right\}_{i \in [n], k \in [t+1]},$$

and similarly for $B_{i,k}$ using F_ℓ instead of F_0.

Phase III Decoding: Recall the definition of $\mathbf{G}(X, Y)$ from Phase I. For the m-th batch, we will use the parenthesized superscript (m). Party P_ℓ needs $\sum_m \omega^{m-1} * \mathbf{G}_\ell^{(m)}(0, \zeta_j)$ to validate party P_j (from Phase II). It can obtain this quantity from $\sum_m \omega^{m-1} * F_\ell^{(m,k)}(X)$ (with evaluations given at each $X = \zeta_i$, but it needs to confirm which of these values, i.e.

$B_{i,k}$ sent by P_i, are correct) and then combining over k using powers of ζ_j. To confirm party P_i, for each k, P_ℓ checks if

$$A_{i,k} + \theta_\ell * B_{i,k} = \sum_{m=1}^{n} \omega^{m-1} * \mathsf{mac}_{\ell,i}^{(m,k)}. \tag{11}$$

With this check, P_ℓ can decode and get $\sum_m \omega^{m-1} * F_\ell^{(m,k)}(X)$, and hence also $\sum_m \omega^{m-1} * F_\ell^{(m,k)}(0)$ and then get $\Psi_{\ell,j} \stackrel{\text{def}}{=} \sum_m \omega^{m-1} * \mathbf{G}_\ell^{(m)}(0,\zeta_j)$ as

$$\Psi_{\ell,j} := \sum_{k=1}^{t+1} \sum_{m=1}^{n} \zeta_j^{k-1} \omega^{m-1} * F_\ell^{(m,k)}(0)$$

This can be used along with the quantities $\mathbf{G}_0^{(m)}(0,\zeta_j)$ received from P_j in Phase II to verify which party P_j (from phase II) is valid, by checking

$$\left(\sum_m \omega^{m-1} * \mathbf{G}_0^{(m)}(0,\zeta_j)\right) + \theta_\ell * \Psi_{\ell,j} == \sum_{m=1}^{n} \sum_{k=1}^{t+1} \omega^{m-1} \zeta_j^{k-1} * \mathsf{mac}_{\ell,*}^{(m,k)} \tag{12}$$

where $\mathsf{mac}_{\ell,*}^{(m,k)}$ is as defined in Eqn. 1.
Thus, with the above check, party P_ℓ can determine which of the parties P_j sent all correct $\{\mathbf{G}_0^{(m)}(0,\zeta_j)\}_m$. With any $t+1$ of these, it can get $\{\mathbf{G}_0^{(m)}(0,Y)\}_m$, and hence all $\{a^{(m,k)}\}_{m\in[n],k\in[t+1]}$.

Proof. The proof of the theorem follows from the following two claims. Claim 1 below shows that for each $m \in [n]$, at the end of Phase I, each honest party P_j has $\mathbf{G}_0^{(m)}(0,\zeta_j)$ except with probability upper bounded by $t/(|\mathbb{E}|-1)$. Claim 2 shows that if honest parties P_j send their decoding from Phase I to P_ℓ, then by end of Phase III, P_ℓ would have decoded $\{a^{(m,k)}\}_{m\in[n],k\in[t+1]}$ (except with probability $n*t/(|\mathbb{E}|-1)$). Now, the maximum number of dishonest parties is at most t. The probability that one party (an honest party) fails to decode Phase I properly across all batches is at most $(t+1)*t*n/(|\mathbb{E}|-1)$ (by union bound and Claim 1). Thus, by union bound, the probability that P_ℓ fails to decode all values is at most $(t+2)*t*n/(|\mathbb{E}|-1)$.

Recall, Phase I and Phase II are run in parallel for the different n batches indexed by m.

Claim 1. For each $m \in [n]$, at the end of Phase I decoding, each honest party P_j has $\mathbf{G}_0^{(m)}(0,\zeta_j)$ except with probability upper bounded by $t/(|\mathbb{E}|-1)$.

Proof (of Claim 1): Suppose in Phase I, a malicious party P_i sends to P_j: $\mathbf{G}_0(\zeta_i,\zeta_j)+\epsilon_0$, and $\mathbf{G}_j(\zeta_i,\zeta_j)+\epsilon_1$, where $\epsilon_0,\epsilon_1 \in \mathbb{E}$. Then the test that P_j conducts

on P_i is equivalent to

$$\begin{aligned}\mathbf{G}_0(\zeta_i,\zeta_j)+\epsilon_0+\theta_j*(\mathbf{G}_j(\zeta_i,\zeta_j)+\epsilon_1) ==& \sum_k \mathsf{mac}_{j,i}^{(k)}*\zeta_j^{k-1}\\ =&\sum_k (F_0^{(k)}(\zeta_i)+\theta_j*F_i^{(k)}(\zeta_i))*\zeta_j^{k-1}\\ =&G_0(\zeta_i,\zeta_j)+\theta_j*G_j(\zeta_i,\zeta_j)\end{aligned}$$

Thus, the test passes only if $\epsilon_0+\theta_j*\epsilon_1=0$. Now, if ϵ_1 is zero then test passes only if ϵ_0 is also zero. If ϵ_1 is not-zero then the test passes only if $\theta_j=\epsilon_0/\epsilon_1$. Since θ_j is independently random of the view of all malicious parties, so is it also independent of ϵ_0/ϵ_1. Thus, the probability above of $\epsilon_1\neq 0$ while the test passes is at most $1/(|\mathbb{E}|-1)$. By union bound, over all dishonest parties, the probability that a dishonest party passes the test while it gave wrong values is at most $t/(|\mathbb{E}|-1)$. Excluding this possibility, P_j can use any $t+1$ parties that pass the test and invert the corresponding vandermonde matrix to obtain $\mathbf{G}_0(X,\zeta_j)$ and hence $\mathbf{G}_0(0,\zeta_j)$.

Claim 2. At the end of Phase III decoding, each honest party P_ℓ has $a^{(m,k)}$ for all m in $[n]$ and all k in $[t+1]$, except with probability upper-bounded by $n*t/(|\mathbb{E}|-1)$.

Proof: We prove this claim in two parts. We first prove that for each party P_ℓ, its test (11) of P_i's material received in Phase III passes only if the received material is indeed $\{F_0^{(k)}(\zeta_i)\}_{k\in[t+1]}$ and $\{F_\ell^{(k)}(\zeta_i)\}_{j\in[n],k\in[t+1]}$, except for probability $(t+1)/(|\mathbb{E}|-1)$. To prove this, for each party P_ℓ, for all i,k, lets first define the valid $\bar{A}_{i,k}$ and $\bar{B}_{i,k}$ it should receive if party P_i were honest (in Phase III sending).

$$\left\{\bar{A}_{i,k}\stackrel{\text{def}}{=}\sum_{m=1}^{n}\omega^{m-1}*F_0^{(m,k)}(\zeta_i)\right\}_{\substack{i\in[n]\\k\in[t+1]}},\left\{\bar{B}_{i,k}\stackrel{\text{def}}{=}\sum_{m=1}^{n}\omega^{m-1}*F_\ell^{(m,k)}(\zeta_i)\right\}_{\substack{i\in[n]\\k\in[t+1]}}$$

It is easy to check that

$$\bar{A}_{i,k}+\theta_\ell*\bar{B}_{i,k}=\sum_{m=1}^{n}\omega^{m-1}*\mathsf{mac}_{\ell,i}^{(m,k)},$$

by definition of $\mathsf{mac}_{\ell,i}^{(m,k)}$, so that test (11) passes if P_i is honest. Now, suppose Party P_i is dishonest and sends $\langle A_{i,k},B_{i,k}\rangle$ different from $\langle\bar{A}_{i,k},\bar{B}_{i,k}\rangle$. Note that *both* these pairs are independent of θ_l, the secret key of honest party P_ℓ, since the view of the Adversary is independent of θ_ℓ and the honest $\langle\bar{A}_{i,k},\bar{B}_{i,k}\rangle$ is chosen independent of θ_ℓ. By lemma 3.2, since θ_ℓ is chosen at random from $\mathbb{E}$, this different pair, $\langle A_{i,k},B_{i,k}\rangle$, cannot pass this test with probability higher than $1/(|\mathbb{E}|-1)$. Thus, with high probability $A_{i,k}=\bar{A}_{i,k}$ and $B_{i,k}=\bar{B}_{i,k}$ if the test passes.

Thus, this allows P_ℓ to use any $t+1$ parties that pass the test (11) to decode $\{\sum_{m=1}^{n}\omega^{m-1}*F_\ell^{(m,k)}(X)\}_{k\in[t+1]}$ and hence also $\{\sum_{m=1}^{n}\omega^{m-1}*F_\ell^{(m,k)}(0)\}_{k\in[t+1]}$.

From this, party P_ℓ computes $\sum_k \zeta_j^{k-1} \sum_m \omega^{m-1} * F_\ell^{(m,k)}(0) = \sum_{m=1}^{n} \omega^{m-1} * \mathbf{G}_\ell^{(m)}(0, \zeta_j)$, which is denoted $\Psi_{\ell,j}$ in the opening protocol (Phase III decoding).

Next, we show that test (12) passes only if party P_j in Phase II sent $\mathbf{G}_0^{(m)}(0, \zeta_j)$ correctly for all $m \in [n]$. Suppose some dishonest party P_j sent to P_ℓ in Phase II : $\mathbf{G}_0^{(m)}(0, \zeta_j) + \epsilon_m$.

Now test (12) in Phase III is as follows:

$$\left(\sum_m \omega^{m-1} * (\mathbf{G}_0^{(m)}(0, \zeta_j) + \epsilon_m)\right) + \theta_\ell * \Psi_{\ell,j} == \sum_{m=1}^{n} \sum_{k=1}^{t+1} \omega^{m-1} \zeta_j^{k-1} * \mathsf{mac}_{\ell,*}^{(m,k)}$$
$$== \sum_{m=1}^{n} \sum_{k=1}^{t+1} \omega^{m-1} \zeta_j^{k-1} * (F_0^{(m,k)}(0) + \theta_\ell * F_\ell^{(m,k)}(0))$$

by definition of $\mathsf{mac}_{\ell,*}^{(m,k)}$ in Fig 1. Rewriting this test, by collecting terms corresponding to each power of ω, and using definition of $\Psi_{\ell,j}$, the test passes iff

$$\sum_{m=1}^{n} \omega^{m-1} * \Big(\mathbf{G}_0^{(m)}(0, \zeta_j) + \epsilon_m + \theta_\ell * \mathbf{G}_\ell^{(m)}(0, \zeta_j) - \sum_k \zeta_j^{k-1} * (F_0^{(m,k)}(0) + \theta_\ell * F_\ell^{(m,k)}(0))\Big) = 0$$

Since the quantities in the large parentheses were either defined independent of ω or chosen by Adversary (i.e. ϵ_m) before ω was opened (and with honest majority privacy of robust-encoding), by Schwartz-Zippel lemma, the probability that any of the coefficients of powers of ω above are non-zero can happen with probability at most $(n-1)/(|\mathbb{E}|-1)$. Thus, with probability at least $1-(n-1)/(|\mathbb{E}|-1)$, we have, for all $m \in [n]$,

$$\mathbf{G}_0^{(m)}(0, \zeta_j) + \epsilon_m + \theta_\ell * \mathbf{G}_\ell^{(m)}(0, \zeta_j) \; - \; \sum_k \zeta_j^{k-1} * (F_0^{(m,k)}(0) + \theta_\ell * F_\ell^{(m,k)}(0)) \; = 0$$

By definition of $\mathbf{G}(X, Y)$ it follows that all ϵ_m are zero, i.e. party P_j sent all $\mathbf{G}_0^{(m)}(0, \zeta_j)$ correctly. Thus, using any $t+1$ parties that pass the test (12), P_ℓ can decode from these $\mathbf{G}_0^{(m)}(0, \zeta_j)$ the degree t polynomial $\mathbf{G}_0^{(m)}(0, Y)$ (for each m separately), and hence all $\{a^{(m,k)}\}_{m\in[n],k\in[t+1]}$.

Note that the probability that one of the dishonest party passes the test (11) is at most $t/(|\mathbb{E}|-1)$ (by union bound), and similarly by union bound and Schwartz-Zippel lemma used above, the probability that some dishonest party manages to pass the aggregate test (12) is at most $(n-1) * t/(|\mathbb{E}|-1)$. Thus, by union bound, the adversary's probability of going undetected is at most $n * t/(|\mathbb{E}|-1)$. Otherwise, party P_ℓ decodes correctly.

C Deferred Proofs of Section 5

C.1 Proof of Lemma 5.2

Proof. For efficient decoding, unfold the triples into a scalar word of length $N_{\text{out}} = 3n$. Every member of $\mathcal{L}(\boldsymbol{y})$ has at least $T := 3(t+1)$ scalar agreements.

The Guruswami–Sudan algorithm [41] finds every polynomial of degree $< W$ having more than $\sqrt{N_{\text{out}}(W-1)}$ agreements, and

$$\begin{aligned} T^2 - N_{\text{out}}(W-1) &\geq 9(t+1)^2 - (6t+3)t \\ &= 3t^2 + 15t + 9 > 0. \end{aligned}$$

Thus, run Guruswami–Sudan on the unfolded word and retain precisely the output polynomials agreeing at all three coordinates of at least $t+1$ blocks. The filtering step returns exactly $\mathcal{L}(\boldsymbol{y})$ in polynomial time.

A specialized Sudan alternative. Our parameters also admit a direct multiplicity-one interpolation decoder of the classical Sudan type [52]. Give X weight 1 and Y weight t, and seek a nonzero polynomial

$$Q(X,Y) = A_0(X) + A_1(X)Y + A_2(X)Y^2$$

of Y-degree at most two and weighted degree at most $3t+1$. Equivalently,

$$\deg A_0 \leq 3t+1, \qquad \deg A_1 \leq 2t+1, \qquad \deg A_2 \leq t+1.$$

There are $(3t+2) + (2t+2) + (t+2) = 6t+6$ coefficients and only $3n = 6t+3$ homogeneous constraints

$$Q(\beta_{j,u}, y_{j,u}) = 0 \qquad (j \in [n],\ u \in [3]),$$

so a nonzero solution always exists. For every $g \in \mathcal{L}(\boldsymbol{y})$, the polynomial $Q(X, g(X))$ has degree at most $3t+1$ and at least $3(t+1) = 3t+3$ distinct roots. Hence it is identically zero, or equivalently $Y - g(X)$ divides $Q(X,Y)$. Since Q has Y-degree at most two, factoring it and applying the complete-block filter returns at most two candidates.

C.2 Proof of Lemma 5.3

Proof. Write $\widehat{X}_u = X_{i,j,u} + \delta_u$ and put $z = \theta_j^{-1}$. It follows that $\widehat{Y}_u = Y_{i,j,u} - z\delta_u$. Thus, acceptance implies

$$p(z) := \Phi_3(Y_{i,j,1} - z\delta_1, Y_{i,j,2} - z\delta_2, Y_{i,j,3} - z\delta_3) - \widehat{S} = 0.$$

Let u_0 be the largest index for which $\delta_{u_0} \neq 0$. Terms with index greater than u_0 are constant in z, terms with smaller index have degree less than u_0, and the term with index u_0 has leading coefficient $(-\delta_{u_0})^{u_0}$. Hence p is a nonzero polynomial of degree at most three and has at most three roots in $\mathbb{E} \setminus \{0\}$. Inversion preserves the uniform distribution on $\mathbb{E} \setminus \{0\}$, which proves the claim.

C.3 Proof of Thm. 5.4

Proof. First, assume that no mac authentication was forged. Every honest party P_j then accepts all $n - t \geq t + 1$ honest senders in round 1 and accepts no incorrect sender. It therefore interpolates every value $c_{j,u}$ correctly. For each honest party P_ℓ, the $n - t$ honest parties supply $n - t$ complete, correct blocks. Hence $G_0(0, Y) \in \mathcal{L}_\ell$, regardless of how the t corrupt parties equivocate among receivers. Lemma 5.1 gives $|\mathcal{L}_\ell| \leq 2$. If the list is a singleton, it is therefore $\{G_0(0, Y)\}$. If it has two members, the point ρ_ℓ chosen in Round 3 separates them. Thus in either case, the authenticated value $G_0(0, \rho_\ell)$ identifies $G_0(0, Y)$ deterministically. If no mac authentication was forged, every honest party's Round 2 message is correct and independent of its verification key. Consequently, an honest party P_j's list and deterministic separating point ρ_j introduce no additional information about θ_j beyond the rejected key guesses already accounted for by conditioning.

For authentication, each of the at most $t(n-t)$ packets from a corrupt sender to an honest owner contributes a nonzero key polynomial of degree at most three by Lemma 5.3. Each of the at most $t(n - t)$ false final responses contributes a nonzero polynomial of degree at most one in the corresponding hidden key. By a union bound, the probability of a successful authentication forgery is upper bounded by $\frac{4t(n-t)}{|\mathbb{E}|-1}$. If there is no authentication forgery, every honest P_j reconstructs the correct value $z_j = G_0(0, \rho_j)$ in Round 4 and hence selects $G_0(0, Y)$ uniquely.

Finally, per ordered pair of distinct parties, Round 1 communicates three elements of $\mathbb{F}$ and one of $\mathbb{E}$, Round 2 communicates three elements of $\mathbb{F}$, Round 3 communicates one element of $\mathbb{F}$, and Round 4 communicates one element of $\mathbb{F}$ and one of $\mathbb{E}$. This gives the total communication cost as stated in the theorem statement.

D Ideal Functionality for Extra-Correct SFE

We define an ideal functionality for secure function evaluation (SFE) in the multi-client distributed-server setting. Secure function evaluation has been studied extensively and various UC definitions exist covering even Byzantine and adaptive settings and feasibility of such definitions have been considered in both synchronous and asynchronous communication models (see [16] and references therein for these definitions). In this work, we consider variations of these definitions which can lead to much more efficient realizations, especially in an offline-online setting. The variations add some new features and exclude some features normally considered. The exclusions are mostly for ease of exposition, and the results presented here can mostly be extended to cover these exclusions as well. In particular, we consider boosting the correctness threshold to $n/2$ for a functionality which usually provides security against $n/3$ malicious parties. The ideal functionality $\mathcal{F}_{\mathsf{X\text{-}SFE}[\mathbb{F},f,n,m]}$ is defined in Figure 3.

Some exclusions worth mentioning are (a) *probabilistic* function evaluation, (b) different users getting different functions' evaluations (of the same inputs)

Ideal Functionality $\mathcal{F}_{\text{X-SFE}[\mathbb{F},f,n,m]}$

Participants: A maximum of m users P_i ($i \in [m]$), a global coordinator GC, and adversary $\mathcal{S}$ interact with functionality $\mathcal{F}_{\text{X-SFE}}$.

Local Variables: inputCommitted (Boolean, initialized: false), numServersCorrupted (Integer, initialized: zero, max: n), array M of size m initialized to $\perp$.

Input Phase: On receiving $(\texttt{inp}, i, x)$ from P_i: If numServersCorrupted $\in [n/3, n]$ then send $(\texttt{inp}, i, x)$ to $\mathcal{S}$ else send $(\texttt{inp}, i)$ to $\mathcal{S}$. Next, if inputCommitted then ignore else set $M[i] = x$.

startCompute: On receiving message `startCompute` from GC:

- If numServersCorrupted $\in [0, n/3 - 1]$ then set inputCommitted to true, and compute $y = f(M)$ and output y to GC *and* $\mathcal{S}$ and exit.
- If numServersCorrupted $\in [n/3, n]$ then send `startCompute` to $\mathcal{S}$. When $\mathcal{S}$ responds back with (`output`, y') set inputCommitted to true. Compute $y = f(M)$, and
  - If numServersCorrupted $\in [n/3, n/2 - 1]$: If $y' = y$ then output y to GC and exit *else* output $\perp$ to GC and exit.
  - If numServersCorrupted $\in [n/2, n]$ output y' to GC and exit.

Corruption:

- On receiving a message $(\texttt{corruptOneMoreServer}, i)$ from $\mathcal{S}$ increment numServersCorrupted. If numServersCorrupted $\in [n/3, n]$ then give (current) M to $\mathcal{S}$. If numServersCorrupted $> n/2$, then $\mathcal{S}$ can alter M to M' (before `startCompute`), as well direct to send $\perp$ to GC.
- If $\mathcal{S}$ corrupts a user P_i, then send $M[i]$ to $\mathcal{S}$. Also, if $\mathcal{S}$ provides a x' (including $\perp$) and inputCommitted is false, then set $M[i] = x'$.

Fig. 3: An ideal functionality for multi-client distributed-server extra-robust secure function evaluation (X-SFE) for function f over $\mathbb{F}$ with at most m inputs. The functionality captures the security and privacy requirements of computing f on inputs of m clients using n servers. The global coordinator GC can, for instance, be a global clock (for issuing `startCompute`) and a public bulletin-board for publishing the result.

[guaranteed output delivery of this setting can typically be only realized in the synchronous communication model (see [37, 9])], and (c) peer-to-peer SFE.

E Input Sharing

E.1 Ideal Functionality for Robust Secret-Sharing

We now describe the corresponding ideal functionality $\mathcal{F}_{\text{RSS-PRE}}$ that gives out shares to the n servers.

Ideal Functionality $\mathcal{F}_{\text{RSS-PRE}}[\mathbb{F}, \mathbb{E}, [\,\cdot\,]]$

Parameters: A field $\mathbb{F}$, a finite extension field $\mathbb{E}$ of $\mathbb{F}$, and a d degree Shamir secret-sharing scheme $[\cdot]$ over $\mathbb{F}$.

Participants: An Originator O and n servers $S_1, ..., S_n$.

Generate/Expand: On receiving message (Generate) or (Expand, ${}^d[a]$) from O, do the following:

1. If the originator did not provide an ${}^d[a]$ (i.e. called with (Generate)), then set $a \xleftarrow{\$} \mathbb{F}$ and run the code of the robust secret-sharing scheme $[\![\cdot]\!]^{\mathbb{F},\mathbb{E},n,d}$ with input $a \in \mathbb{F}$ and output $[\![a]\!]_i$ to each server S_i.
2. If the originator provided ${}^d[a]$ (i.e. called with (Expand)), then run the code of $[\![\cdot]\!]^{\mathbb{F},\mathbb{E},n,d}$ with the initial shares $[a]$ (the $F_0(\zeta_j)$'s) set to those provided by the originator and output $[\![a]\!]_i$ to each server S_i

E.2 Ideal Functionality for Input Sharing

We first define an ideal functionality for a client P to secret share its input amongst n servers. We will assume that the client P and the n servers have identities known to each other, as well as secure point-to-point communication channels. Typically, this is known in the literature as the verifiable secret sharing (VSS) ideal functionality ([31, 12]) for which a universally-composable definition is formulated in [17]. However, we want a formulation that is efficiently realizable even when the robustness and correctness thresholds are $n/2$ corruptions whereas the sharing is only $n/3$ private; recall the shares are used in the MPC protocol that is $n/3$ private, but $n/2$ robust and correct.

This definition is motivated by likely scenarios where a set of servers are more incentivized to cause a wrong output (and hence more likely to be corruptible for this end) than to just learn the value being shared. Similarly, there are situations where a client is more concerned about its shared value y being misconstrued as $y' \neq y$ than y being leaked. Of course, there are situations where this is not the case, but our justification is that in the preprocessing model, this higher level of robustness and correctness can be realized with small extra effort.

In fact, we can define three different thresholds for privacy (t_p), robustness (t_r) and correctness (t_c). To reiterate, the security and privacy guarantees can be informally summarized as follows:

Privacy: For $t < t_p$, t malicious (corrupt) servers obtain no information about y, the value shared by honest P.

Robustness: For $t < t_r$, even if t servers are maliciously faulty/corrupt, the other honest servers can reconstruct y, the value shared by honest P.

Client Verifiability: For $t < t_c$, if P is corrupt and t servers are corrupt (and/or colluding with P), then either P distributes $[z]$ for some z or all honest servers declare P corrupt, i.e. *disqualify* P.

Correctness: For $t < t_c$, if P is honest, then t malicious servers cannot cause honest servers to open (reconstruct) to $y' \neq y$, where y is the value shared by honest P, *nor* can these malicious servers cause honest servers to disqualify P.

Ideal Functionality $\mathcal{F}_{\mathsf{inputShare}[\mathbb{F},n,\langle t_p,t_r,t_c\rangle,[\,\cdot\,]]}$

Parameters: A field $\mathbb{F}$, the number of servers n, thresholds t_p, t_r, and t_c, and a sharing scheme $[\,\cdot\,]$ for n parties.

Participants: A client (dealer) P, n servers S_k ($k \in [n]$), and an adversary $\mathcal{S}$ interact with functionality $\mathcal{F}_{\mathsf{inputShare}}$.

Local Variables: numServersCorrupted (Integer, Initialized: 0, Max: n), y (Initialized: $\perp$), $[y]$ (Initialized: $\perp^n$), PCorrupt (Boolean, Initialized: false).

Input: On receiving $(\texttt{inp}, P, x)$ for $x \in \mathbb{F}$ from P: If numServersCorrupted $\geq t_p$ or PCorrupt, send $(\texttt{inp}, P, x)$ to $\mathcal{S}$. Else, send $(\texttt{inp}, P)$ to $\mathcal{S}$. Set $y = x$. Generate a fresh random $[y]$ and give $[y]_i$ to $\mathcal{S}$ for $i \in$ [numServersCorrupted].

Share: On receiving message (Share, y') from $\mathcal{S}$ for $y' \in \mathbb{F} \cup \{\perp\}$,

- If PCorrupt and $y' = \perp$, output $\perp$ to all honest servers.
- Else if numServersCorrupted $< t_r$: Output $[y]_i$ to each honest server S_i and ok to P.
- Else if $t_c >$ numServersCorrupted $\geq t_r$: If $y' = y$, then output $[y]_i$ to each honest server S_i and ok to P. If $y' \neq y$, then output the message "tooManyServersCorrupted" to all honest servers and P.
- Else if numServersCorrupted $\geq t_c$: Output to each honest server and P as dictated by $\mathcal{S}$.

Corruption: On receiving message corruptOneMoreServer from $\mathcal{S}$, increment numServersCorrupted and give $[y]_{\text{numServersCorrupted}}$ to $\mathcal{S}$. If numServersCorrupted $\geq t_p$, give y to $\mathcal{S}$. On receiving message (corruptUser, P) from $\mathcal{S}$, set PCorrupt = true, and give y to $\mathcal{S}$. If P is already corrupted, $\mathcal{S}$ can also send a message (reset, P, z) in which case $\mathcal{F}_{\mathsf{inputShare}}$ updates $[y]_i \leftarrow [y]_i + z - y$, sets $y = z$, and gives $[y]_i$ to $\mathcal{S}$ for $i \in$ [numServersCorrupted].

Fig. 4: An ideal functionality for resilient and verifiable input-sharing of a client's input amongst n servers using general sharing $[\,\cdot\,]$ and privacy (t_p), robustness (t_r), and correctness (t_c) thresholds. The session IDs have been suppressed for readability.

While the above properties are informally described, these notions are formalized in the ideal functionality $\mathcal{F}_{\mathsf{inputShare}}$ described in Fig 4. As opposed to having an "opening" phase as in the UC-definition of VSS in [17], this functionality is parameterized by a secret-sharing scheme and actually gives out shares to the specified servers.

E.3 UC Realization of $\mathcal{F}_{\mathsf{inputShare}}$ in the Preprocessing Model

We now show that the ideal functionality $\mathcal{F}_{\mathsf{inputShare}}$ can be realized for any $[\cdot]$ that is a Shamir secret-sharing scheme using degree $d-1$ polynomials for $d = t_p \leq t_r = t_c \leq n/2$ in the preprocessing model. Importantly, the preprocessing material is only given to the n servers and *not* the client(s). In other words, the client(s) is not involved at all in the preprocessing, and the client(s) does not come into the picture until the actual time of input submission. Also, this protocol is a *single round protocol*, where the client gets information from each of the servers and then the client replies back to the servers. We remark that this protocol is similar to the input sharing scheme of SPDZ [27, Fig 1], except that here the client(s) is not part of the set of parties which have preprocessed data. Yet, if for each input providing client, there was a separate authenticator generated using preprocessing, servers can just send their respective authenticating key to this particular client.

In more detail, this scheme utilizes the robust secret-sharing scheme described in Section 4. Let $\mathcal{F}_{\mathsf{BB}}[t]$ be a broadcast ideal functionality from a client P to n servers which can tolerate up to $t-1$ Byzantine malicious servers[5]. We now describe the protocol $\Pi_{\mathsf{inputShare}}$ realizing $\mathcal{F}_{\mathsf{inputShare}}$ in the $\mathcal{F}_{\mathsf{RSS\text{-}PRE}}$ and $\mathcal{F}_{\mathsf{BB}}$-hybrid model.

Protocol $\Pi_{\mathsf{inputShare}}[\mathbb{F}, n, \langle t_p, t_r, t_c\rangle, [\cdot]]$ using $\mathcal{F}_{\mathsf{RSS\text{-}PRE}}[\mathbb{F}, \mathbb{E}, [\cdot]]$ and $\mathcal{F}_{\mathsf{BB}}[t_c]$

Participants: A client P and n servers S_k ($k \in [n]$).

Input: P receives its input $x \in \mathbb{F}$ from the environment.

Server Preprocessing: An originator invokes $\mathcal{F}_{\mathsf{RSS\text{-}PRE}}[\mathbb{F}, \mathbb{E}, [\cdot]]$. This results in the n servers getting populated by correlated randomness $[\![\rho]\!]$ for a random $\rho \in \mathbb{F}$. More precisely, server S_i has $F_0(\zeta_i)$, for a polynomial $F_0(X)$ whose constant term is ρ. S_i also has $\{F_j(\zeta_i)\}_{j\in[n]}$, a and $\mathsf{mac}_{i.j}$ (for all $j \in [n]$). It also has θ_i.

Server S_i: Server S_i sends $[\![\rho]\!]_i$ to P.

P Verifies and Submits: P receives from every S_i a secret θ_i^*, $n+1$ values $\{F_j^{*,i}\}_{j\in[0..n]}$, and n macs $\mathsf{mac}_{i,j}^*$ For each i, P now emulates S_i as in $\Pi_{\mathsf{RSS}}.\mathtt{OpenSound}$'s verification and reconstruction step. Each such emulation leads to either server S_i rejecting (output $\perp$) or outputting a

[5]Note that such a broadcast functionality can be realized with the client sending its message once to the other parties (servers), and from there on only servers are involved in a Byzantine consensus protocol.

$\rho^{(i)}$. If a majority of (emulated) S_i's $\rho^{(i)}$ are equal to some ρ, then P uses $\mathcal{F}_{\mathsf{BB}}[t_c]$ to (Byzantine) broadcast $x - \rho$ to the n servers. If not (i.e. no ρ reached majority), then P aborts.

Server S_i: On receiving some u in the broadcast from P, output $[\rho]_i + u$ as its share.

Theorem E.1. *For $d = t_p \le t_r = t_c \le n/2$ where $[\cdot]$ is a degree $d-1$ Shamir secret-sharing scheme, the above protocol $\Pi_{\mathsf{inputShare}}[\mathbb{F}, n, \langle t_p, t_r, t_c\rangle, [\,\cdot\,]]$ UC-realizes $\mathcal{F}_{\mathsf{inputShare}}[\mathbb{F}, n, \langle t_p, t_r, t_c\rangle, [\,\cdot\,]]$ in the $\mathcal{F}_{\mathsf{RSS\text{-}PRE}}[\mathbb{F}, \mathbb{E}, [\,\cdot\,]]$, $\mathcal{F}_{\mathsf{BB}}[t_c]$-hybrid model with statistical distance $\le (d-1)/|\mathbb{E}|$ as long as the total number of servers corrupted is less than t_c.*

We first give a description of the simulator $\mathcal{S}$ that interacts with the ideal functionality $\mathcal{F}_{\mathsf{inputShare}}$. We will use $\mathcal{F}$ to refer to $\mathcal{F}_{\mathsf{inputShare}}$ and Π to refer to $\Pi_{\mathsf{inputShare}}$ for simplicity of notation. WLOG, we will assume $\mathcal{A}$ corrupts the first $\mathsf{numServersCorrupted}$ servers by index and use $t = \mathsf{numServersCorrupted}$. Note that $t < t_c$.

Simulator $\mathcal{S}$

For every server S_i corrupted by $\mathcal{A}$, send `corruptOneMoreServer` to $\mathcal{F}$. We handle simulation in three cases.

P Honest, $t \ge t_p$:

- $\mathcal{S}$ receives x from $\mathcal{F}$ along with $[x]_i$ for $i \in [t]$. Since $t \ge t_p$, it can compute $[x]_i$ for all $i \in [n]$.
- $\mathcal{S}$ samples a random $a \in \mathbb{F}$, sets $[y]_i = a + [x]_i$ for all $i \in [n]$ and runs the code of $\mathcal{F}_{\mathsf{RSS\text{-}PRE}}[\mathbb{F}, \mathbb{E}, [\cdot]]$ with (Expand, $[y]$) to obtain $[\![y]\!]$. For each $i \in [t]$, it sends $[\![y]\!]_i$ to S_i.
- When P broadcasts in Π, $\mathcal{S}$ sends $(\texttt{Share}, x)$ to $\mathcal{F}$ and runs the code of $\mathcal{F}_{\mathsf{BB}}[t_c]$ to broadcast $-a$ to the servers corrupted by $\mathcal{A}$.

P Honest, $t < t_p$:

- $\mathcal{S}$ receives $[x]_i$ for $i \in [t]$ from $\mathcal{F}$.
- $\mathcal{S}$ samples a random $a \in \mathbb{F}$ and sets $[y]_i = a + [x]_i$ for all $i \in [t]$. For all $i \in [n] \backslash [t]$, it sets $[y]_i$ at random conditioned on them being consistent with the already known $[y]_i$'s for $i \in [t]$. $\mathcal{S}$ runs the code of $\mathcal{F}_{\mathsf{RSS\text{-}PRE}}[\mathbb{F}, \mathbb{E}, [\cdot]]$ with (Expand, $[y]$) to obtain $[\![y]\!]$. For each $i \in [t]$, it sends $[\![y]\!]_i$ to S_i.
- When P broadcasts in Π, $\mathcal{S}$ sends $(\texttt{Share}, 0)$ to $\mathcal{F}$ and runs the code of $\mathcal{F}_{\mathsf{BB}}[t_c]$ to broadcast $-a$ to the servers corrupted by $\mathcal{A}$.

P Corrupt:

- $\mathcal{S}$ sends $(\texttt{corruptUser}, P)$ to $\mathcal{F}$ and receives x along with $[x]_i$ for $i \in [t]$ from $\mathcal{F}$.
- For each $i \in [n] \backslash [t]$, $\mathcal{S}$ samples consistent shares $[x]_i$ with x and the already known $[x]_j$ for $j \in [t]$. If $t \ge t_p$, these additional shares for the honest servers are already completely determined. For $t < t_p$, the linear system of equations is underdetermined, and $\mathcal{S}$ satisfies

it by choosing a solution uniformly at random and then computing the additional shares.
- $\mathcal{S}$ samples a random $a \in \mathbb{F}$ and sets $[y]_i = a + [x]_i$ for all $i \in [n]$.
- $\mathcal{S}$ runs the code of $\mathcal{F}_{\mathsf{RSS\text{-}PRE}}[\mathbb{F}, \mathbb{E}, [\cdot]]$ with (Expand, $[y]$) to obtain $[\![y]\!]$. It emulates Π by sending $[\![y]\!]_i$ for each $i \in [t]$ to S_i and for each $i \in [n] \backslash [t]$ to P.
- If P runs $\mathcal{F}_{\mathsf{BB}}[t_c]$ with value z, $\mathcal{S}$ checks if $z = -a$. If so, $\mathcal{S}$ sends $(\texttt{Share}, x)$ to $\mathcal{F}$. Otherwise, $\mathcal{S}$ sends $(\mathsf{reset}, P, z + a + x)$ and then $(\texttt{Share}, z + a + x)$ to $\mathcal{F}$.
- Else if P aborts, $\mathcal{S}$ sends $(\texttt{Share}, \perp)$ to $\mathcal{F}$.

W.l.o.g., we will assume that the adversary $\mathcal{A}$ in the UC protocol $\Pi_{\mathsf{inputShare}}$ is a dummy adversary and essentially passes back and forth commands and messages from the environment $\mathcal{E}$. Thus, from now on, we will use the environment $\mathcal{E}$ as the real adversary. We will show that the view of $\mathcal{E}$ interacting with $\mathcal{F}_{\mathsf{inputShare}}$ and the simulator $\mathcal{S}$ described above is statistically indistinguishable from the view of $\mathcal{E}$ interacting with $\Pi_{\mathsf{inputShare}}$. We first show the following lemma.

Lemma E.2. *When P is honest and $t < t_c \leq n/2$, in the real world execution of $\Pi_{\mathsf{inputShare}}$, P will abort with probability $\leq (d-1)/|\mathbb{E}|$.*

Proof. This follows immediately from correctness of robust secret-sharing which means that if at least $d = t_p$ servers are honest, then all honest servers output a except with probability $\leq (d-1)/|\mathbb{E}|$. Since $t_c \leq n/2$, this implies $n - t_c \geq n/2 \geq t_p$. This means that in P's emulation, all honest servers output a except with probability $\leq (d-1)/|\mathbb{E}|$. Since $t < n/2$, a majority of servers are honest and output a, and so P does not abort except with probability $\leq (d-1)/|\mathbb{E}|$.

We now handle each corruption case separately.

Case P Honest, $t \geq t_p$: For uniform $a \in \mathbb{F}$, a and $x + a$ are identically distributed, so $\mathcal{E}$'s view up to pre-processing step is identical in the real and ideal worlds. In the real world protocol ρ is to be seen as a replacement for a used by the Simulator. Similarly, $x - a$ and $-a$ are identically distributed. In Π, P's verification check fails with probability $\leq (d-1)/|\mathbb{E}|$, so we don't have perfect simulation. P's broadcast and resulting shares of all parties are identically distributed by the same argument. So, $\mathcal{E}$'s view is identical in the real and ideal worlds as long as P's verification check passes.

Case P Honest, $t < t_p$: For uniform $a \in \mathbb{F}$, a and $x+a$ are identically distributed. Moreover, since $t < t_p$, $[y]_i$ is information-theoretically hidden, so $\mathcal{E}$'s view up to pre-processing step is identical in the real and ideal worlds. In Π, P's verification check fails with probability $\leq (d-1)/|\mathbb{E}|$, so we don't have perfect simulation. P's broadcast and resulting shares of all parties are identically distributed by the same argument. So, $\mathcal{E}$'s view is identical in the real and ideal worlds as long as P's verification check passes.

Case P Corrupt: For uniform $a \in \mathbb{F}$, a and $x + a$ are identically distributed, and the ideal functionality's generation of $[x]_i$'s are identically distributed to the simulator's when filling in remaining shares. So, all the pre-processing material given to $\mathcal{E}$ is identically distributed in the real and ideal worlds. P's broadcast then determines the output of all servers, which are identically distributed in the real world to the ideal functionality regardless of choice. Thus, $\mathcal{E}$'s view is identically distributed in the real and ideal worlds.

Thus, in all cases, the statistical distance between $\mathcal{E}$'s view in the real and ideal worlds is $\leq (d-1)/|\mathbb{E}|$.

F Ideal Functionality $\mathcal{F}_{\mathsf{Beaver}}$

The ideal functionality $\mathcal{F}_{\mathsf{Beaver}}$ is described in detail in Figure 5.

Ideal Functionality $\mathcal{F}_{\mathsf{Beaver}}[\mathbb{F}, \mathbb{E}, n, \ell, c]$

Qualification: Let $\mathbb{E}$ be a finite extension of $\mathbb{F}$ with $|\mathbb{E}| > (\ell + c) \cdot n^2 \cdot 2^\lambda$, where λ is the statistical security parameter.

Participants: An originator O and n servers $S_1, ..., S_n$.

Generate: On receiving message (Generate) from O, $\mathcal{F}$ does the following:

1. Pick n evaluation points $\theta_i \in \mathbb{E}$ ($i \in [n]$) uniformly and independently at random, conditioned on all n points being non-zero and distinct. Output θ_i to S_i. Call this array of points $\vec{\theta}$.
2. For $k = 1$ to ℓ do:
   i. Sample $\rho_k \xleftarrow{\$} \mathbb{F}$.
   ii. Generate fresh random $\langle \rho_k \rangle$ (note: as per our default convention, these are degree $n/2 - 1$ Shamir shares).
   iii. Call $\mathcal{F}_{\mathsf{RSS\text{-}PRE}}$ with (Expand, $\langle \rho_k \rangle$) to get $\langle\!\langle \rho_k \rangle\!\rangle_{\mathsf{inp}}$ along with its evaluation points w_i.
   iv. Generate fresh random $\langle\!\langle \rho_k \rangle\!\rangle$ using the evaluation points $\vec{\theta}$.
   v. Distribute $(\langle\!\langle \rho_k \rangle\!\rangle_{\mathsf{inp}}, \langle\!\langle \rho_k \rangle\!\rangle)$ (share-wise) to the n servers.
3. For $k = 1$ to c do:
   i. Pick $l_k, r_k \xleftarrow{\$} \mathbb{F}$.
   ii. Generate fresh random $\langle\!\langle l_k \rangle\!\rangle$, $\langle\!\langle r_k \rangle\!\rangle$, $\langle\!\langle l_k r_k \rangle\!\rangle$, all of these using the same evaluation points $\vec{\theta}$. Distribute these (share-wise) to the n servers.

Fig. 5: Beaver Multiplication Tuples

G Ideal Functionality X-Beaver

Ideal Functionality $\mathcal{F}_{\mathsf{X\text{-}Beaver}}[\mathbb{F}, \mathbb{E}, n, \ell, c]$

Qualification: Let $\mathbb{E}$ be a finite extension of $\mathbb{F}$ with $|\mathbb{E}| > (\ell + c) \cdot n^2 \cdot 2^\lambda$, where λ is the statistical security parameter.

Participants: An originator O and n servers $S_1, ..., S_n$.

Generate: On receiving message (Generate) from O, $\mathcal{F}$ does the following:

1. Pick n evaluation points $\theta_i \in \mathbb{E}$ ($i \in [n]$) uniformly and independently at random, conditioned on all n points being non-zero and distinct. Output θ_i to S_i. Call this array of points $\vec{\theta}$.
2. Sample random non-zero $\alpha, x \xleftarrow{\$} \mathbb{E}$. Generate fresh random $\langle\!\langle \alpha \rangle\!\rangle^{\mathbb{E}}$ and $\langle\!\langle x \rangle\!\rangle^{\mathbb{E}}$ using evaluation points $\vec{\theta}$ and distribute them (share-wise) to the n servers.
3. For $k = 1$ to ℓ do:
   i. Sample $\rho_k \xleftarrow{\$} \mathbb{F}$.
   ii. Generate fresh random $[\rho_k]$ (note: as per our default convention, these are degree $n/3 - 1$ Shamir shares).
   iii. Call $\mathcal{F}_{\mathsf{RSS\text{-}PRE}}$ with (Expand, $[\rho_k]$) to get $[\![\rho_k]\!]^{n,n/3}$ along with its evaluation points w_i (note: $[\![\rho_k]\!]^{n,n/3}_i$ contains w_i).
   iv. Generate fresh random $\langle\!\langle \rho_k \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$ using the evaluation points $\vec{\theta}$.
   v. Distribute $([\![\rho_k]\!]^{n,n/3}, \langle\!\langle \rho_k \cdot \alpha \rangle\!\rangle^{\mathbb{E}})$ (share-wise) to the n servers. This tuple will be referred to as an **input-converting authenticated tuple**.
4. For $k = 1$ to c do:
   i. Pick $l_k, r_k \xleftarrow{\$} \mathbb{F}$.
   ii. Generate fresh random $[l_k], [r_k]$, and $[l_k r_k]$ and distribute them (share-wise) to the n servers (note: as per our default convention, these are degree $n/3 - 1$ Shamir shares).
   iii. Generate fresh random $\langle\!\langle l_k \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$, $\langle\!\langle r_k \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$, $\langle\!\langle l_k r_k \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$, all of these using the same evaluation points $\vec{\theta}$. Distribute these (share-wise) to the n servers.
   iv. The complete set $[l_k], [r_k], [l_k r_k], \langle\!\langle l_k \cdot \alpha \rangle\!\rangle^{\mathbb{E}}, \langle\!\langle r_k \cdot \alpha \rangle\!\rangle^{\mathbb{E}}, \langle\!\langle l_k r_k \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$ will be referred to as an **extended and authenticated Beaver tuple**. The first part, i.e. $[l_k], [r_k], [l_k r_k]$, will be referred to as a **basic Beaver tuple**.
5. Pick a random n-vector $\vec{\beta} = \langle \beta_1, ..., \beta_n \rangle$ in $\mathbb{E}$. Modify the robust secret-sharing scheme described in Figure 1, where β_i is used instead of $F_0(\zeta_i)$. That is, $\vec{\beta}$ replaces the evaluations of $F_0(X)$. The resulting codeword, generated using the evaluation points $\vec{\theta}$, will be written as $\langle\!\langle \vec{\beta} \rangle\!\rangle^{\mathbb{E}}$. Distribute the shares of this codeword to the n servers.

H Proof of Theorem 9.1 (ONLINE – X-SFE)

Theorem 9.1 (repeated) *The protocol $\Pi_{\mathsf{ONLINE-X\text{-}SFE}[\mathbb{F},f,n,m]}$ UC-realizes ideal functionality $\mathcal{F}_{\mathsf{X\text{-}SFE}[\mathbb{F},f,n,m]}$ in the $\mathcal{F}_{\mathsf{X\text{-}Beaver}}$, $\mathcal{F}_{\mathsf{inputShare}}$, $\mathcal{F}_{\mathsf{BC}}$-hybrid model, where $\mathcal{F}_{\mathsf{BC}}$ denotes the ideal functionality for honest-majority Byzantine consensus.*

Proof. We will assume that the adversary $\mathcal{A}$ in the UC protocol Π is dummy, and essentially passes back and forth commands and messages from the environment $\mathcal{E}$. Thus, from now on we will use environment $\mathcal{E}$ as the real adversary, which outputs a single bit. The simulator $\mathcal{S}$ will also be the ideal world adversary for $\mathcal{F}_{\mathsf{X\text{-}SFE}}$. It is a universal simulator that uses $\mathcal{A}$ as a black box.

We note that the description of $\Pi_{\mathsf{ONLINE-X\text{-}SFE}}$ uses $\Pi_{\mathsf{inputShare}}$ instead of $\mathcal{F}_{\mathsf{inputShare}}$. This is because $\Pi_{\mathsf{inputShare}}$ implements $\mathcal{F}_{\mathsf{inputShare}}$ in the $\mathcal{F}_{\mathsf{RSS\text{-}PRE}}$-hybrid model (see Lemma E.1). By the UC-composition theorem we could replace $\Pi_{\mathsf{inputShare}}$ by $\mathcal{F}_{\mathsf{inputShare}}$ in the description of $\Pi_{\mathsf{ONLINE-X\text{-}SFE}}$, except that $\Pi_{\mathsf{ONLINE-X\text{-}SFE}}$ itself uses pre-processing functionality $\mathcal{F}_{\mathsf{X\text{-}Beaver}}$. Since $\mathcal{F}_{\mathsf{X\text{-}Beaver}}$ and $\mathcal{F}_{\mathsf{RSS\text{-}PRE}}$ use a common state, in particular a global (random) MAC key α and (random) evaluation points $\vec{\theta}$, one has to use the JUC-composition theorem, i.e the joint-sessions universal composability theorem proved in [18]. So, from now we will assume that $\Pi_{\mathsf{ONLINE-X\text{-}SFE}}$ is a $\mathcal{F}_{\mathsf{X\text{-}Beaver}}$, $\mathcal{F}_{\mathsf{inputShare}}$,$\mathcal{F}_{\mathsf{BC}}$-hybrid protocol with joint state $\alpha, \vec{\theta}$.

The simulator $\mathcal{S}$ is described next, which takes $\mathcal{A}$ as a blackbox, and interacting with ideal functionality $\mathcal{F}_{\mathsf{X\text{-}SFE}}$ (as participant $\mathcal{S}$) simulates the view of $\mathcal{A}$ as in the real world. The main idea of the simulator is that it will emulate the ideal functionality $\mathcal{F}_{\mathsf{X\text{-}Beaver}}$ and simulate $\mathcal{F}_{\mathsf{inputShare}}$ as well as all of the honest servers in Π. Note that the parties in the real world protocol Π are a maximum of m clients P_j, a GC, and the n servers. However, since the m clients give their input shares using $\mathcal{F}_{\mathsf{inputShare}}$, we will assume that the clients P_j ($j \in [m]$) are not participants of Π, but the n servers are pre-populated with shares of the $\ell \le m$ inputs of the ℓ clients that succeeded in input phase. In other words, assuming there were ℓ valid inputs from clients ($\ell \le m$), their inputs a_j are given to the n servers as $\hat{a}_j = a_j + \rho_j, [a_j], \langle\!\langle \rho_j \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$.

Simulator $\mathcal{S}$

The simulator $\mathcal{S}$ emulates the code of ideal functionality $\mathcal{F}_{\mathsf{X\text{-}Beaver}}$, while also simulating the ℓ clients' shares $\hat{a}_j = a_j + \rho_j, [a_j], \langle\!\langle \rho_j \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$ (for $j \in 1..\ell$). We describe the simulation in the static corruption model, where the number and identity of corrupted servers is known at the outset. Suppose the set of servers corrupted is $\mathcal{C}$ and let $n_{\mathsf{bad}} = |\mathcal{C}|$. $\mathcal{S}$ calls $\mathcal{F}_{\mathsf{X\text{-}SFE}}$ n_{bad} times with `corruptOneMoreServer`, so that $\mathcal{F}_{\mathsf{X\text{-}SFE}}$ has $\mathsf{numServersCorrupted} = n_{\mathsf{bad}}$.

Case $n_{\mathsf{bad}} < n/3$. The simulator does not get the actual inputs M, but just the number of participants ℓ that have valid inputs (see the input phase of $\mathcal{F}_{\mathsf{X\text{-}SFE}}$). So, it just runs the code of $\mathcal{F}_{\mathsf{X\text{-}Beaver}}$ and also generates shares for

the ℓ clients with all ℓ inputs set to zero. It gives the resulting data of the first n_{bad} servers to $\mathcal{A}$.

Next, it calls $\mathcal{F}.\texttt{startCompute}$ and gets the output $y = f(M)$. Thereafter, the simulator executes the same code as Π for the honest servers treating $\mathcal{A}$ as a black-box for messages to and from corrupted servers except for one change: at step vii of the Attestation Phase, the simulator gives shares of the honest servers for the opening of $\langle\!\langle l \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$ (the l used in the final gate) so that these are consistent with the opening of $(l - f(M) + f(\vec{0})) \cdot \alpha$, as described in Lemma H.1.

Case $n_{\mathsf{bad}} \geq n/3$. In this case, the ideal functionality $\mathcal{F}_{\mathsf{X\text{-}SFE}}$ gives $\mathcal{S}$ the actual inputs M. The simulator then uses these actual inputs M instead of zero in the previous case of $n_{\mathsf{bad}} < n/3$. Thus, the simulator executes the same code as Π for the honest servers treating $\mathcal{A}$ as a black-box for messages to and from corrupted servers. At the end of the protocol, all honest servers output some value y_j in the simulation. $\mathcal{S}$ also gets from $\mathcal{A}$ the values it is outputting for each corrupt server. If there is a value y' that has frequency $\geq n/2$, the simulator calls $\mathcal{F}_{\mathsf{X\text{-}SFE}}$ with message $(\texttt{output}, y')$. Else, the simulator calls $\mathcal{F}_{\mathsf{X\text{-}SFE}}$ with message $(\texttt{output}, \perp)$.

This finishes the description of the simulator.

Lemma H.1. *The simulator $\mathcal{S}$ which chooses the global authentication key α, the public evaluation points $\vec{\zeta}$ (for the $\langle\cdot\rangle$ secret-sharing) and $\vec{\theta}$ (for the $\langle\!\langle\cdot\rangle\!\rangle$ secret-sharing), as well as random l for the final gate, and has access to only $f(M)$ where M is the clients' input, can efficiently compute an opening of the honest parties in attestation step vii so that it is consistent with an opening of $\langle\!\langle (l - f(M) + f(\vec{0})) \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$ instead of an opening of $\langle\!\langle l \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$ in the view of the adversary that has static-corrupted $< n/3$ servers.*

Proof. W.l.o.g., assume that servers $1..n_{\mathsf{bad}}$ are corrupt. Also, let $\vec{r}$ be the n_{bad}-shares of $\langle l \cdot \alpha \rangle$ given to $\mathcal{A}$, which are the values $F_0(\zeta_i)$ (for a polynomial with $F_0(0) = \ell \cdot \alpha$, given to $\mathcal{A}$ ($i \in [1..n_{\mathsf{bad}}], j \neq i$). Let $\overline{\mathbf{V}}$ be the top $n/2$ rows of Vandermonde matrix $\mathbf{V}^{n \times n/2}$ (using evaluation points $\zeta_1, ..., \zeta_n$). Let $\bar{v}^\top$ be the top row of inverse of $\overline{\mathbf{V}}$ (invertible, because it is Vandermonde). Then, letting $\vec{s}$ stand for the $n/2 - n_{\mathsf{bad}}$ shares of the as many honest servers, we need

$$\vec{v}^\top \cdot \begin{pmatrix} \vec{r} \\ \vec{s} \end{pmatrix} = (l - f(M) + f(\vec{0})) \cdot \alpha$$

Since $n_{\mathsf{bad}} < n/3 < n/2 - 1$, $\vec{s}$ has rank at least one. The simulator can use any one element from this (affine) subspace for $\vec{s}$, which together with $\vec{r}$, then determine all other shares of $\langle l \cdot \alpha \rangle$.

$\mathcal{S}$ also needs to give $F_0(\zeta_i), F_j(\zeta_i)$ for all $i \in [n_{\mathsf{bad}} + 1..n], j \neq i$, along with $\mathsf{mac}_{i,j}$ for all $i \in [n_{\mathsf{bad}}+1..n], j \neq i$. Recall, $\mathcal{A}$ already has $F_0(\zeta_i), F_j(\zeta_i)$ and $\mathsf{mac}_{i,j}$ for $i \in [1..n_{\mathsf{bad}}], j \neq i$. Let sh_i denote the ith (simulated) share of $\langle l \cdot \alpha \rangle$ previously determined by $\mathcal{S}$ and v^*_{ji} denote the evaluations $\mathsf{mac}_{i,j}$. For $i \in [n_{\mathsf{bad}} + 1..n], j \in$

$[1..n_{\mathsf{bad}}]$, $\mathcal{S}$ determines b_{ij} such that $\mathsf{sh}_i + b_{ij}\theta_j = v^*_{ij}$ and sets $F_j(\zeta_i) = b_{ij}$. It similarly sets $v^*_{ij} = F_0(\zeta_i) + F_j(\zeta_i) * \theta_j$.

We now describe a series of experiments between a probabilistic polynomial time challenger Chal and the environment $\mathcal{E}$, starting with Expt_0 which we describe next. We will show that the view of $\mathcal{E}$ in Expt_0 is same as its view in the ideal-world setting with $\mathcal{E}$ interacting with $\mathcal{F}_{\mathsf{X\text{-}SFE}}$ and the simulator $\mathcal{S}$ described above. We end with an experiment which is identical to the real world execution of the protocol in Π. We will show that the environment $\mathcal{E}$ has negligible advantage in distinguishing between these series of experiments, leading to a proof of realization of $\mathcal{F}_{\mathsf{X\text{-}SFE}}$ by the protocol $\Pi_{\mathsf{X\text{-}SFE}}$.
Here is the complete code of Expt_0 (stated as it's overall experiment with $\mathcal{E}$).

Expt_0**:**

(i) The challenger Chal gets from $\mathcal{E}$ the number of participants $\ell \le m$ plus their (array of) inputs M. It computes and saves $y = f(M)$. Next, if $n_{\mathsf{bad}} < n/3$ then it sets the input array M to all zero.

(ii) The challenger Chal then runs the code $\mathcal{F}_{\mathsf{X\text{-}Beaver}}$ as well as generates consistent input shares $\hat{a}_j = a_j + \rho_j, [a_j], \langle\!\langle \rho_j \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$ of each element a_j in the array M. It designates servers $\mathcal{C} = [1..n_{\mathsf{bad}}]$ as corrupt and $[n_{\mathsf{bad}} + 1..n]$ as honest. For each tuple generated in $\mathcal{F}_{\mathsf{X\text{-}Beaver}}$ as well as input shares, it sends the first n_{bad} shares to $\mathcal{E}$. It also sends the first n_{bad} secret evaluation points θ_i (generated in code of $\mathcal{F}_{\mathsf{X\text{-}Beaver}}$) to $\mathcal{E}$.

(iii) Chal next emulates protocol Π for the honest servers with $\mathcal{E}$ running the part of the corrupt servers except for the following change at the end of emulation.

$n_{\mathsf{bad}} < n/3$**:** In the last step where $l \cdot \alpha$ is opened, i.e. Chal sends shares of $l \cdot \alpha$ of the honest servers to $\mathcal{E}$, Chal re-calculates the shares of honest servers differently so that the opening will be to $l - y + f(\vec{0})$ instead. This is possible as described in detail above in the description of $\mathcal{S}$. Chal also outputs y to $\mathcal{E}$.

$n_{\mathsf{bad}} \ge n/3$**:** At the end of the protocol, all honest servers output some value y_j at the end of the simulation. Chal also gets from $\mathcal{E}$ the values it is outputting for each corrupt server. If there is a value y' that has frequency $\ge n/2$, let $y^* = y'$. Else, let $y^* = \perp$. Chal outputs y^* to $\mathcal{E}$.

It is straightforward to inspect that the view of $\mathcal{E}$ in Expt_0 is identical to its view in its combined interaction with $\mathcal{F}_{\mathsf{X\text{-}SFE}}$ and simulator $\mathcal{S}$, as Chal has just combined the code of $\mathcal{F}_{\mathsf{X\text{-}SFE}}$ and $\mathcal{S}$.

Expt_1:

This experiment differs from Expt_0 in step (i) in that the array M is not reset to zero for the $n_{\mathsf{bad}} < n/3$ case, *and*, in step (iii), again for the $n_{\mathsf{bad}} < n/3$ case, the opening of $l \cdot \alpha$ is not adjusted and done just as in real protocol Π.

Lemma H.2. [Privacy] *The experiments* Expt_0 *and* Expt_1 *are identically distributed for* $\mathcal{E}$*.*

Proof. First note that for a random $l{\cdot}\alpha$ (which is independent of y (= $f(M)$)): the pairs $\langle f(\vec{0}){+}l{\cdot}\alpha, l{\cdot}\alpha{-}y{+}f(\vec{0})\rangle$ and $\langle y{+}l{\cdot}\alpha, l{\cdot}\alpha\rangle$ are identically distributed for any y. Since $< n/3$ shares of a (probabilistic) encoding ${}^{d}[x]$ of x, for $d \geq n/3$, do not reveal any information about x, the experiments Expt_0 and Expt_1 are distributed identically for $\mathcal{E}$ by Lemma H.1. Note that in both these experiments, in the $n_{\mathsf{bad}} < n/3$ case, the game aborts if the honest-majority Byzantine consensus output is $\perp$.

Expt_2: This experiment is the same as the real world, i.e. Chal runs the honest servers in $\Pi_{\mathsf{ONLINE\text{-}X\text{-}SFE}}$ against $\mathcal{E}$.

The proof is complete by the following two lemmas H.3 and H.4, which together show that the statistical distance between Expt_1 and Expt_2 is negligible.

Lemma H.3. [Robustness] *When* $n_{\mathsf{bad}} < n/3$*, every honest server's input to the honest-majority Byzantine consensus in step viii of the attestation phase is* $f(M)$ *with overwhelming probability.*

Proof. To start with, by correctness of robust secret-sharing scheme, since the number of honest servers is more than $2n/3$, honest servers will correctly decode x, α, and $l{\cdot}\alpha$ with overwhelming probability. Further, since only $< n/3$ servers are corrupt, an honest server's computation of A and B in attestation step ii and A's eventual decoding in step iv will be correct with overwhelming probability. This follows since at every multiplication gate and the output wire, the values $a_l + l$ and $a_r + r$ obtained using $\mathsf{OpenRobust}$ will be correct. Therefore, every honest server's computation of each $\hat{a}$ and l will be correct, and it follows that, with overwhelming probability, every honest server's input to the honest-majority Byzantine consensus is $f(M)$.

Lemma H.4. [Correctness] *When* $n_{\mathsf{bad}} \in [n/3, n/2-1]$*, if an honest server's input to the honest-majority Byzantine consensus in step viii of the attestation phase is not* $\perp$*, then with overwhelming probability it is* $f(M)$*.*

Proof. If an honest server's input to the honest-majority Byzantine consensus in the attestation phase is not $\perp$, then the check $A \stackrel{?}{=} B{\cdot}\alpha$ passed for this server. We will show that if the check $A \stackrel{?}{=} B{\cdot}\alpha$ passes, then with overwhelming probability, the $\hat{a}$ value of the output obtained by this server is correct (meaning it is the $\hat{a}$ value that would be obtained during an honest execution of the protocol. By the correctness of $\Pi_{\mathsf{RSS}}.\mathtt{OpenSound}$, it follows that α and $l \cdot \alpha$ are correctly opened with overwhelming probability, and this server's input to the honest-majority Byzantine consensus is $\hat{a} - l = f(M)$.

We now show that if the check $A \stackrel{?}{=} B \cdot \alpha$ passes, then the $\hat{a}$ value obtained by this server S_i is correct with overwhelming probability. We first observe that since $n_{\mathsf{bad}} < n/2$, x, α, and $\vec{\beta}$ are opened correctly with overwhelming probability.

Let A_{hon} and B_{hon} denote the values of A and B that would be obtained during the attestation phase of the honest protocol execution. Let A_i and B_i denote the values actually obtained by S_i. Let J denote the set of servers $j \in [n]$ that S_i uses shares from to reconstruct A_i. S_i will use its own share, so note that $i \in J$. Let $A^{(j)}$ denote the jth share of A obtained by S_j during the protocol execution. Let $A^{(j)}_{\mathsf{hon}}$ denote the values of these shares during the honest execution. It follows for reconstruction coefficients λ_j that

$$A_i = \sum_{j \in J} \lambda_j A^{(j)}$$

and

$$A_{\mathsf{hon}} = \sum_{j \in J} \lambda_j A^{(j)}_{\mathsf{hon}}.$$

With overwhelming probability, there is no authentication forgery when opening A_i or A_{hon}, so all shares accepted by an honest server S_i lie on the same receiver-relative polynomial $F_0^{(i)}(Z, X)$ determined formally before x is opened. Therefore, every successful reconstruction yields the same $A_i = F_0^{(i)}(0, x)$ independently of J and its interpolation coefficients. Thus, the set J and coefficients $\{\lambda_j\}$ which are chosen after x is opened can be replaced in the following analysis by a fixed canonical representation $F_0^{(i)}(0, X)$ chosen before x is opened. We present it instead using a fixed interpolation set J for ease of exposition.

Let $J_{\mathsf{hon}} \subseteq J$ be the set of honest servers in J and let $J_{\mathsf{corr}} \subseteq J$ be the set of corrupted servers in J. For honest servers $j \in J_{\mathsf{hon}}$, we can similarly define $P_g^{(j)}$ and $Q_g^{(j)}$ as the shares of P_g and Q_g, respectively, used by S_j to compute $A^{(j)}$. For all servers $j \in J$, we define $P^{(j)}_{g,\mathsf{hon}}$ and $Q^{(j)}_{g,\mathsf{hon}}$ analogously as the shares obtained during the honest protocol execution. For $j \in J_{\mathsf{hon}}$, define $P_g^{(j)} = P^{(j)}_{g,\mathsf{hon}} + \delta_g^{(j)}$ and $Q_g^{(j)} = Q^{(j)}_{g,\mathsf{hon}} + \epsilon_g^{(j)}$. For $j \in J_{\mathsf{corr}}$, define $\delta_g^{(j)} = \epsilon_g^{(j)} = 0$. It follows that

$$A_i = \sum_{j \in J_{\mathsf{corr}}} \lambda_j \Delta^{(j)} + \sum_{j \in J} \lambda_j \sum_{g=\ell+1}^{\ell+|C|} x^{2(g-\ell)}(P^{(j)}_{g,\mathsf{hon}} + \delta_g^{(j)}) + x^{2(g-\ell)+1}(Q^{(j)}_{g,\mathsf{hon}} + \epsilon_g^{(j)}),$$

where $\Delta^{(j)}$ denotes the arbitrary difference of the jth share determined by the adversary (note that this models the fact that the adversarial $\Delta^{(j)}$ may not be a low degree polynomial in x). Observe that $\Delta^{(j)}$ is independent of α. This follows since the view of the adversary when it sends v_i thus committing to its share $A^{(j)}$ is independent of α since the honest parties' shares are masked by β_i's. Define $R^{(i)}_{g,\mathsf{hon}}$ and $S^{(i)}_{g,\mathsf{hon}}$ as the values of R_g and S_g that S_i would obtain in the honest execution of the protocol, and $R_g^{(i)}$ and $S_g^{(i)}$ as the values actually obtained by server S_i. Define $R_g^{(i)} = R^{(i)}_{g,\mathsf{hon}} + \tau_g^{(i)}$ and $S_g^{(i)} = S^{(i)}_{g,\mathsf{hon}} + \psi_g^{(i)}$. Observe that the $\tau_g^{(i)}$'s and $\psi_g^{(i)}$'s are independent of α since they can be determined by the adversary whose view during the protocol execution is independent of α. Furthermore, note that the $\delta_g^{(j)}$'s and $\epsilon_g^{(j)}$'s can be expressed in the form $c \cdot \alpha$

for some c independent of α. This follows by noting that any error introduced in computing $P_g^{(j)}$ is determined by the computed value of $\langle\!\langle \rho_l \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$, which is determined by the openings $a_{l'} + l'$ and $a_{r'} + r'$ (l' and r' denote the left and right indices of the gate with output l) which are known to the adversary whose view is independent of α. Define $\delta_g'^{(j)}$ (resp. $\epsilon_g'^{(j)}$) as $\delta_g^{(j)}/\alpha$ (resp. $\epsilon_g^{(j)}/\alpha$).

Suppose $A_i = B_i \cdot \alpha$. Then,

$$\sum_{j \in J_{\mathsf{corr}}} \lambda_j \Delta^{(j)} + \sum_{j \in J} \lambda_j \sum_{g=\ell+1}^{\ell+|C|} x^{2(g-\ell)} (P_{g,\mathsf{hon}}^{(j)} + \delta_g^{(j)}) + x^{2(g-\ell)+1} (Q_{g,\mathsf{hon}}^{(j)} + \epsilon_g^{(j)}) = B_i \cdot \alpha,$$

which implies

$$\sum_{j \in J_{\mathsf{corr}}} \lambda_j \Delta^{(j)} = \alpha \cdot \left[\sum_{g=\ell+1}^{\ell+|C|} x^{2(g-\ell)} (\tau_g^{(i)} - \sum_{j \in J_{\mathsf{hon}}} \lambda_j \delta_g'^{(j)}) + x^{2(g-\ell)+1} (\psi_g^{(i)} - \sum_{j \in J_{\mathsf{hon}}} \lambda_j \epsilon_g'^{(j)}) \right].$$

The above expression is of the form $W = \alpha \cdot V$ with both W and V independent of α. Thus, with overwhelming probability, $W = V = 0$. By the Schwartz-Zippel lemma, it follows that with overwhelming probability,

$$\sum_{g=\ell+1}^{\ell+|C|} X^{2(g-\ell)} (\tau_g^{(i)} - \sum_{j \in J_{\mathsf{hon}}} \lambda_j \delta_g'^{(j)}) + X^{2(g-\ell)+1} (\psi_g^{(i)} - \sum_{j \in J_{\mathsf{hon}}} \lambda_j \epsilon_g'^{(j)}) = 0$$

as a polynomial and thus all coefficients are 0. This holds because, although the interpolation set J is determined after x is opened, accepted-set invariance allows us to use the canonical representation described above, whose coefficients are fixed before x is opened and are therefore independent of x. Thus,

$$\forall g \in [\ell+1..\ell+|C|]\,,\ \tau_g^{(i)} - \sum_{j \in J_{\mathsf{hon}}} \lambda_j \delta_g'^{(j)} = 0 \text{ and } \psi_g^{(i)} - \sum_{j \in J_{\mathsf{hon}}} \lambda_j \epsilon_g'^{(j)} = 0. \quad (13)$$

We now show inductively that all the errors are 0 and thus $A_i = A_{\mathsf{hon}}$ and $B_i = B_{\mathsf{hon}}$. First, due to the correctness of the input phase (Lemma E.1), it follows that honest servers have the correct shares $([a], \hat{a}, \langle\!\langle \rho \cdot \alpha \rangle\!\rangle^{\mathbb{E}})$ for input wires. By correct shares, we mean that these shares match the values of those in the honest execution of the protocol. It is immediate that if all input shares to an addition gate are correct, then the shares of the output gate are also correct. We induct sequentially over the multiplication gates. For the base case, consider the first multiplication gate $g = \ell + 1$. Since the input wires are correct, it follows that the inputs to this multiplication gate g are correct. Thus, $P_g^{(j)}$ and $Q_g^{(j)}$ are correct and so $\delta_g'^{(j)}$ and $\epsilon_g'^{(j)}$ are 0 for all $j \in J_{\mathsf{hon}}$. By (13), this implies $\tau_g^{(i)} = \psi_g^{(i)} = 0$, which implies that $a_l + l$ and $a_r + r$ were opened correctly and thus, $([a_g], \hat{a}_g, \langle\!\langle \rho_g \cdot \alpha \rangle\!\rangle^{\mathbb{E}})$ are correctly computed by server S_i.

Now consider any multiplication gate $g \in [\ell + 1..\ell + |C|]$. By the induction hypothesis, S_i computed the correct shares for the inputs to this multiplication gate. It immediately follows that $P_g^{(i)}$ and $Q_g^{(i)}$ are computed correctly by S_i and

so $\delta_g'^{(i)} = \epsilon_g'^{(i)} = 0$. For any $j \in J_{\mathsf{hon}}$, since S_i accepted $A^{(j)}$ as consistent with its share $A^{(i)}$, it follows from Lemma 4.4 that $P_g^{(j)}$ and $Q_g^{(j)}$ are consistent with $P_g^{(i)}$ and $Q_g^{(i)}$, respectively. Therefore, by Lemma 4.3, $\delta_g'^{(j)} = \epsilon_g'^{(j)} = 0$ for all $j \in J_{\mathsf{hon}}$ with overwhelming probability. By (13), this implies $\tau_g^{(i)} = \psi_g^{(i)} = 0$, which implies that $a_l + l$ and $a_r + r$ were opened correctly and thus, $([a_g], \hat{a}_g, \langle\!\langle \rho_g \cdot \alpha \rangle\!\rangle^{\mathbb{E}})$ are correctly computed by server S_i.

Thus, for all $g \in [\ell + 1..\ell + |C|]$, $\tau_g^{(i)} = \psi_g^{(i)} = 0$, which implies that the entire computation is computed correctly by S_i, and the output $\hat{a}$ obtained by server S_i is correct with overwhelming probability.

Lemma H.4 immediately implies that if the honest-majority Byzantine consensus in the attestation phase is a value $y \neq \perp$, then $y = f(M)$.

I Extension of Thm. 1.3 with Honest-Majority Robustness

In this section, we show how the protocol of Thm. 1.3 can be extended to also achieve honest-majority robustness. In the event that the attestation phase fails, the parties can run the online phase of Thm. 1.2 as a fallback using the same preprocessing material. Since the protocol was already robust against up to $n/3$ corruptions, if the attestation phase fails, it is guaranteed that $\geq n/3$ parties are corrupt. First, observe that since the input phase of Thm. 1.3 satisfies honest-majority robustness, the parties are guaranteed (for each input a_i) to have the value $\hat{a}_i = a_i + \rho_i$ and shares of $[a_i], \langle\!\langle \rho_i \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$. Furthermore, for each multiplication gate, the parties have an extended and authenticated Beaver tuple $[l_k], [r_k]$, $[l_k r_k]$, $\langle\!\langle l_k \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$, $\langle\!\langle r_k \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$, $\langle\!\langle l_k r_k \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$. In order to run the online stage of Thm. 1.2, the parties need to have (for each input a_i) shares of $\langle\!\langle a_i \rangle\!\rangle$ and for each multiplication gate, a Beaver tuple $\langle\!\langle l_k \rangle\!\rangle, \langle\!\langle r_k \rangle\!\rangle, \langle\!\langle l_k r_k \rangle\!\rangle$. During the attestation phase, α was opened, so by the linearity of the robust secret-sharing scheme, all parties can take the $\langle\!\langle l_k \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$, $\langle\!\langle r_k \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$, $\langle\!\langle l_k r_k \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$ portion of their extended and authenticated Beaver tuples and divide by α to obtain Beaver tuples $\langle\!\langle l_k \rangle\!\rangle, \langle\!\langle r_k \rangle\!\rangle, \langle\!\langle l_k r_k \rangle\!\rangle$. Similarly, to obtain $\langle\!\langle a_i \rangle\!\rangle$, the parties can divide $\langle\!\langle \rho_i \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$ by α to obtain $\langle\!\langle \rho_i \rangle\!\rangle^{\mathbb{E}}$ and then subtract this from $\hat{a}_i$ to obtain $\langle\!\langle a_i \rangle\!\rangle$. Since the underlying secret points θ_i and mac tags are still secret after the attestation phase of the protocol of Thm. 1.3, robustness of the online phase of the protocol of Thm. 1.2 holds.

J Preprocessing Analysis for Honest-Majority Protocol (Thm. 1.2)

We give a simple subroutine $\mathsf{NonZeroRand}$ that allows the parties to generate shares of a uniformly random nonzero element θ.

Protocol NonZeroRand

Output: $\langle\theta\rangle$ for a uniformly random nonzero θ.

Protocol: 1. Parties invoke 2 random gates to get $\langle\theta\rangle$ and $\langle r\rangle$.

2. Parties invoke a multiplication gate to get $\langle r\theta\rangle$ and an output gate to open $r\theta$ to all parties. If $r\theta == 0$, the parties discard these shares and repeat until $r\theta \neq 0$.

NonZeroRand uses (with overwhelming probability, since $|\mathbb{E}| > 2^\lambda$) 2 random gates, 1 multiplication gate, and 1 output gate.

We now give the circuit to compute the preprocessing material for our honest-majority protocol for a circuit with up to ℓ inputs and m multiplication gates.

Preprocessing for our Honest-Majority Protocol

Given: Number of multiplication gates m, number of inputs ℓ, and public evaluation points $\zeta_1, \ldots, \zeta_n$.

Output: ℓ input tuples $(\langle\!\langle \rho_k \rangle\!\rangle_{\mathsf{inp}}, \langle\!\langle \rho_k \rangle\!\rangle)_{k\in[\ell]}$, m multiplication tuples $(\langle\!\langle l_k \rangle\!\rangle, \langle\!\langle r_k \rangle\!\rangle, \langle\!\langle l_k r_k \rangle\!\rangle)_{k\in[m]}$ all using the same θ_i's except for the $\langle\!\langle \rho_k \rangle\!\rangle_{\mathsf{inp}}$'s, which all use fresh $\theta_i^{(k)}$'s.

Protocol: 1. Parties invoke NonZeroRand n times to get $\langle\theta_j\rangle$ for $j \in [n]$ and n output gates to output θ_j to P_j.

2. For $k \in [m]$:
   (a) Parties invoke 2 random gates to get $\langle l_k\rangle$ and $\langle r_k\rangle$.
   (b) Parties use a multiplication gate to compute $\langle l_k r_k\rangle$.
   (c) Parties invoke Augment on $\langle l_k\rangle, \langle r_k\rangle, \langle l_k r_k\rangle$ using the $\langle\theta_j\rangle$'s to obtain $\langle\!\langle l_k \rangle\!\rangle, \langle\!\langle r_k \rangle\!\rangle, \langle\!\langle l_k r_k \rangle\!\rangle$.

3. For $k \in [\ell]$:
   (a) Parties invoke a random gate to get $\langle\rho_k\rangle$ and NonZeroRand n times to get $\langle\theta_j^{(k)}\rangle$ for $j \in [n]$ and n output gates to output $\theta_j^{(k)}$ to P_j.
   (b) Parties invoke Augment on $\langle\rho_k\rangle$ twice, once using the $\langle\theta_j^{(k)}\rangle$'s and once using the $\langle\theta_j\rangle$'s to obtain $\langle\!\langle \rho_k \rangle\!\rangle_{\mathsf{inp}}, \langle\!\langle \rho_k \rangle\!\rangle$.

Observe that the preprocessing for our honest-majority protocol requires a circuit of size $O((m+\ell)n^2)$ to compute. Thus, the preprocessing material can be generated in communication $O(n^3|C|) + \mathsf{poly}(n)$ using [39]. Observe that we save a factor n communication compared to the preprocessing generation for [29].

K Preprocessing Analysis for Enhanced-$n/3$ MPC Protocol

K.1 Enhanced-Correctness $n/3$ Protocol

In this subsection, we instantiate $\mathcal{F}_{\mathsf{X\text{-}Beaver}}$, the preprocessing for our $n/3$-private and robust protocol with honest-majority correctness (Thm. 1.3). Observe that

the extended and authenticated Beaver tuples generated by $\mathcal{F}_{\mathsf{X\text{-}Beaver}}$ contain sharings at two different thresholds. The basic Beaver tuple is an $n/3$-sharing, whereas the authentication tuple is an $n/2$-sharing. Thus, to instantiate $\mathcal{F}_{\mathsf{X\text{-}Beaver}}$, it is necessary to convert an $n/2$-sharing to an $n/3$-sharing. We give the subroutine $\mathsf{Clone}_{t,t'}$ to accomplish this.

Protocol $\mathsf{Clone}_{t,t'}$

Given: A sharing ${}^{t}[s]$ employing public evaluation points $\zeta_1, \ldots, \zeta_n$.

Output: A sharing ${}^{t'}[s]$, for $t' \leq t$, using the same evaluation points. Additionally, the parties collectively get t-sharings of each party's share in this new ${}^{t'}[s]$.

Protocol:
1. Parties invoke $t'-1$ random gates to get ${}^{t}[v_i]$, for $i \in [t'-1]$ and each v_i random and independent in $\mathbb{E}$.
2. Parties locally compute (using local additions and scalar multiplications) n t-sharings, $U_j = {}^{t}[s + \sum_{\ell=1}^{t'-1} \zeta_j^{\ell} \cdot v_\ell] = {}^{t}[s] + \sum_{\ell=1}^{t'-1} \zeta_j^{\ell} \cdot {}^{t}[v_\ell]$, for $j \in [n]$.
3. The parties use n output gates to open U_j to P_j for $j \in [n]$, i.e. P_j gets $s + \sum_{\ell=1}^{t'-1} \zeta_j^{\ell} \cdot v_\ell$. Note that this collectively forms $[s]_{t'}$ using the same evaluation points. Also, note the parties already have (in Step 2) a t-sharing of each party's share $s + \sum_{\ell=1}^{t'-1} \zeta_j^{\ell} \cdot v_\ell$. While these additional sharings are not output using an output gate, the correct functioning of the MPC protocol's random gates implies that step 1 was performed correctly, and further the correctness of the output gates implies that these are indeed the sharings of each party's shares in new ${}^{t'}[s]$.

Note that $\mathsf{Clone}_{t,t'}$ uses $t'-1$ random gates and n output gates.

We now give the protocol XBeaver to compute an extended and authenticated Beaver tuple

$$[l], [r], [lr], \langle\!\langle l \cdot \alpha \rangle\!\rangle^{\mathbb{E}}, \langle\!\langle r \cdot \alpha \rangle\!\rangle^{\mathbb{E}}, \langle\!\langle lr \cdot \alpha \rangle\!\rangle^{\mathbb{E}}.$$

Protocol XBeaver

Given: A sharing $\langle \alpha \rangle$ and sharings of secret evaluation points $\langle \theta_1 \rangle, \ldots, \langle \theta_n \rangle$.

Output: An extended and authenticated Beaver tuple $[l], [r], [lr], \langle\!\langle l \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$, $\langle\!\langle r \cdot \alpha \rangle\!\rangle^{\mathbb{E}}, \langle\!\langle lr \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$ using evaluation points $\theta_1, \ldots, \theta_n$.

Protocol:
1. Parties invoke 2 random gates to get $\langle l \rangle$ and $\langle r \rangle$.
2. Parties use a multiplication gate to compute $\langle lr \rangle$.
3. Parties use 3 multiplication gates to compute $\langle l \cdot \alpha \rangle, \langle r \cdot \alpha \rangle, \langle lr \cdot \alpha \rangle$.
4. Parties run Augment on $\langle l \cdot \alpha \rangle, \langle r \cdot \alpha \rangle, \langle lr \cdot \alpha \rangle$ to obtain $\langle\!\langle l \cdot \alpha \rangle\!\rangle^{\mathbb{E}}, \langle\!\langle r \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$, $\langle\!\langle lr \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$.
5. Parties run $\mathsf{Clone}_{n/2,n/3}$ on $\langle l \rangle, \langle r \rangle, \langle lr \rangle$ to obtain $[l], [r], [lr]$.

XBeaver uses 2 random gates, 4 multiplication gates, 3 calls to Augment, and 3 calls to $\mathsf{Clone}_{n/2,n/3}$. This requires a circuit of size $O(n^2)$ as calls to Augment

are the dominant term. We now give the protocol InputTuple to compute an input-converting authenticated tuple $[\![\rho]\!]^{n,n/3}, \langle\!\langle \rho \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$.

Protocol InputTuple

Given: A sharing $\langle \alpha \rangle$ and sharings of secret evaluation points $\langle w_1 \rangle, \ldots, \langle w_n \rangle$ and $\langle \theta_1 \rangle, \ldots, \langle \theta_n \rangle$.

Output: An input-converting authenticated tuple $[\![\rho]\!], \langle\!\langle \rho \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$ with $[\![\rho]\!]$ using the w_i's as evaluation points and $\langle\!\langle \rho \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$ using the θ_i's as evaluation points.

Protocol:
1. Parties invoke a random gate to get $\langle \rho \rangle$.
2. Parties use a multiplication gate to compute $\langle \rho \cdot \alpha \rangle$.
3. Parties run Augment on $\langle \rho \cdot \alpha \rangle$ using the θ_i's as evaluation points to obtain $\langle\!\langle \rho \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$.
4. Parties run $\mathsf{Augment}_{\lceil n/3 \rceil - 1}$ on $\langle \rho \rangle$ using the w_i's as evaluation points to obtain $[\![\rho]\!]$.

InputTuple also requires a circuit of size $O(n^2)$ to compute as Augment is the dominant term.

We now give the instantiation of $\mathcal{F}_{\mathsf{X\text{-}Beaver}}$.

Instantiation of $\mathcal{F}_{\mathsf{X\text{-}Beaver}}$

Given: Number of multiplication gates m, number of inputs ℓ, and public evaluation points $\zeta_1, \ldots, \zeta_n$.

Output: Secrets θ_i to P_i, ℓ input-converting authenticated tuples, m extended and authenticated Beaver tuples, $\langle\!\langle \alpha \rangle\!\rangle^{\mathbb{E}}, \langle\!\langle x \rangle\!\rangle^{\mathbb{E}}, \langle\!\langle \vec{\beta} \rangle\!\rangle^{\mathbb{E}}$.

Protocol:
1. Parties invoke NonZeroRand n times to get $\langle \theta_j \rangle$ for $j \in [n]$.
2. Parties use n output gates to output θ_j to P_j.
3. Parties invoke NonZeroRand twice to get $\langle \alpha \rangle$ and $\langle x \rangle$.
4. Parties invoke Augment twice using the θ_j's as the evaluation points to get $\langle\!\langle \alpha \rangle\!\rangle^{\mathbb{E}}$ and $\langle\!\langle x \rangle\!\rangle^{\mathbb{E}}$.
5. For $k \in [\ell]$:
   (a) Parties invoke NonZeroRand n times to get $\langle w_j^{(k)} \rangle$ for $j \in [n]$.
   (b) Parties use n output gates to output $w_j^{(k)}$ to P_j.
   (c) Parties run InputTuple using $\langle \alpha \rangle$, the $\langle w_j^{(k)} \rangle$'s, and the $\langle \theta_j \rangle$'s to get $[\![\rho_k]\!], \langle\!\langle \rho_k \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$.
6. For $k \in [m]$:
   (a) Parties run XBeaver using $\langle \alpha \rangle$ and the $\langle \theta_j \rangle$'s to get $[l_k], [r_k], [l_k r_k]$, $\langle\!\langle l_k \cdot \alpha \rangle\!\rangle^{\mathbb{E}}, \langle\!\langle r_k \cdot \alpha \rangle\!\rangle^{\mathbb{E}}, \langle\!\langle l_k r_k \cdot \alpha \rangle\!\rangle^{\mathbb{E}}$.
7. Parties use n random gates to get $\langle \beta_1 \rangle, \ldots, \langle \beta_n \rangle$ and n output gates to output β_j to P_j.

8. Parties run $\mathsf{Augment}^*$ on $\langle\beta_1\rangle, \ldots, \langle\beta_n\rangle$ using the $\langle\theta_j\rangle$'s to get $\langle\!\langle\vec{\beta}\rangle\!\rangle^{\mathbb{E}}$, where $\mathsf{Augment}^*$ is $\mathsf{Augment}$, except $\langle\beta_i\rangle$ is used in place of $\langle F_0(\zeta_i)\rangle$.

K.2 Complexity Analysis of $\mathcal{F}_{\mathsf{X\text{-}Beaver}}$ Instantiation

Our instantiation of $\mathcal{F}_{\mathsf{X\text{-}Beaver}}$ uses m calls to $\mathsf{XBeaver}$, ℓ calls to $\mathsf{InputTuple}$, $n+2+n\ell$ calls to $\mathsf{NonZeroRand}$, $2n+n\ell$ output gates, n random gates, 2 calls to $\mathsf{Augment}$, and 1 call to $\mathsf{Augment}^*$. The dominant terms are the calls to $\mathsf{XBeaver}$ and $\mathsf{InputTuple}$, and thus, $\mathcal{F}_{\mathsf{X\text{-}Beaver}}$ can be computed by a circuit of size $O((m+\ell)n^2)$. Therefore, the preprocessing material can be generated in communication $O(n^3|C|) + \mathsf{poly}(n)$ using [39].

L Analysis of Preprocessing Complexity of Robust Shares of [29]

Robust Secret-Sharing Scheme of [29]

- Sample $t+1$ independent univariate random polynomials over $\mathbb{E}$ (each of degree t), $F_0(X), ..., F_t(X)$, with the only constraint being that the constant term of $F_0(X)$ is set to $a \in \mathbb{F}$, *the value being encoded*. We denote this vector of polynomials as $\mathbf{F}(X)$. **Each party P_j gets $\mathbf{F}(\zeta_j)$**, which are the $(t+1)$ evaluations of $\mathbf{F}(X)$ at ζ_j. Note that each value $F_i(\zeta_j)$ is in $\mathbb{E}$.
- Sample $t+1$ independent univariate random polynomials over $\mathbb{E}$ (each of degree t), $A_0(Y), \cdots, A_t(Y)$. We denote this vector of polynomials by $\mathbf{A}(Y)$. **Each party P_j gets $\mathbf{A}(\zeta_j)$**, which are the $(t+1)$ evaluations of $\mathbf{A}(Y)$ at ζ_j. The polynomials $\mathbf{A}(Y)$ can be reused over many different encodings.
- Let $C(X,Y) = \sum_{i=0}^{t} F_i(X) \cdot A_i(Y)$. **Each party P_j gets the coefficients of the polynomial $C(X, \zeta_j)$.**

Fig. 6

We first give a subroutine GenA to generate shares of a vector polynomial $\mathbf{A}(Y)$ and then $\mathsf{EF-Augment}$ to convert a sharing $\langle s\rangle$ to $\langle\!\langle s\rangle\!\rangle$ using a fixed $\mathbf{A}(Y)$ polynomial, where $\langle\!\langle\cdot\rangle\!\rangle$ refers to the robust secret sharing scheme in Fig. 6. Note that $t = \lceil n/2\rceil - 1$.

GenA

Given: Public evaluation points $\zeta_1, \ldots, \zeta_n$.
Output: Sharings $\langle A_i(\zeta_j)\rangle$ for $i \in \{0, \ldots, t\}, j \in [n]$.

Protocol: 1. Parties invoke $(t+1)^2$ random gates to get $\langle A_{ij}\rangle$, for $i,j \in \{0,\ldots,t\}$ with each A_{ij} random and independent in $\mathbb{E}$. Define $A_i(Y) = \sum_{j=0}^{t} A_{ij}Y^j$.
2. Parties locally compute (using public additions and scalar multiplications) $\langle A_i(\zeta_j)\rangle$ for all $i \in \{0,\ldots,t\}$ and $j \in [n]$.
3. Parties use output gates to output $\langle A_i(\zeta_j)\rangle$ to P_j for all $i \in \{0,\ldots,t\}$, $j \in [n]$.

EF – Augment

Given: A sharing $\langle s\rangle$ using public evaluation points $\zeta_1,\ldots,\zeta_n$ and sharings of $\langle A_i(\zeta_j)\rangle$ for $i \in \{0,\ldots,t\}, j \in [n]$.
Output: An augmented sharing $\langle\!\langle s\rangle\!\rangle$ using the $A_i(\zeta_j)$'s.
Protocol: 1. Let $\langle F_{00}\rangle = \langle s\rangle$.
2. Parties invoke $(t+1)^2 - 1$ random gates to get the remaining $\langle F_{ij}\rangle$, for $i,j \in \{0,\ldots,t\}$ with each F_{ij} random and independent in $\mathbb{E}$. Define $F_i(X) = \sum_{j=0}^{t} F_{ij}X^j$.
3. Parties locally compute (using public additions and scalar multiplications) $\langle F_i(\zeta_j)\rangle$ for all i and $j \in [n]$.
4. Let $C(X,Y) = \sum_{i=0}^{t} F_i(X)\cdot A_i(Y)$. Note that for $j \in [n]$, $C(X,\zeta_j) = \sum_{i=0}^{t} F_i(X)\cdot A_i(\zeta_j) = \sum_{i'=0}^{t} X^{i'}(\sum_{i=0}^{t} F_{ii'}\cdot A_i(\zeta_j))$. Each coefficient of $C(X,\zeta_j)$ is computed using $t+1$ multiplication gates (between $\langle F_{ii'}\rangle$ and $\langle A_i(\zeta_j)\rangle$) and local additions.
5. Parties use output gates to output the shares to the appropriate parties (e.g., $\langle F_i(\zeta_j)\rangle$ is opened to P_j).

Observe that EF – Augment requires $(t+1)^2 - 1$ random gates, $n(t+1)^2$ multiplication gates, and $2n(t+1)$ output gates. In total, it requires a circuit of size $O(n^3)$ to compute inside the MPC protocol. We now give the circuit to compute the preprocessing material for [29] for a circuit with up to ℓ inputs and m multiplication gates.

Preprocessing for [29]

Given: Number of multiplication gates m, number of inputs ℓ, and public evaluation points $\zeta_1,\ldots,\zeta_n$.
Output: ℓ input tuples $(\langle\!\langle\rho_k\rangle\!\rangle_{\mathsf{inp}}, \langle\!\langle\rho_k\rangle\!\rangle)_{k\in[\ell]}$, m multiplication tuples $(\langle\!\langle l_k\rangle\!\rangle, \langle\!\langle r_k\rangle\!\rangle, \langle\!\langle l_k r_k\rangle\!\rangle)_{k\in[m]}$ all using the same $\mathbf{A}(Y)$ except for the $\langle\!\langle\rho_k\rangle\!\rangle_{\mathsf{inp}}$'s, which all use fresh $\mathbf{A}(Y)$'s.
Protocol: 1. Parties invoke GenA to obtain $\langle A_i(\zeta_j)\rangle$ for $i \in \{0,\ldots,t\}$, $j \in [n]$.
2. For $k \in [m]$:
   (a) Parties invoke 2 random gates to get $\langle l_k\rangle$ and $\langle r_k\rangle$.
   (b) Parties use a multiplication gate to compute $\langle l_k r_k\rangle$.

(c) Parties invoke $\mathsf{EF-Augment}$ on $\langle l_k \rangle, \langle r_k \rangle, \langle l_k r_k \rangle$ using the $\langle A_i(\zeta_j) \rangle$'s to obtain $\langle\!\langle l_k \rangle\!\rangle, \langle\!\langle r_k \rangle\!\rangle, \langle\!\langle l_k r_k \rangle\!\rangle$.

3. For $k \in [\ell]$:
   (a) Parties invoke a random gate to get $\langle \rho_k \rangle$ and GenA to get $\langle A_i^{(k)}(\zeta_j) \rangle$ for $i \in \{0, \ldots, t\}, j \in [n]$.
   (b) Parties invoke $\mathsf{EF-Augment}$ on $\langle \rho_k \rangle$ twice, once using the $\langle A_i^{(k)}(\zeta_j) \rangle$'s and once using the $\langle A_i(\zeta_j) \rangle$'s to obtain $\langle\!\langle \rho_k \rangle\!\rangle_{\mathsf{inp}}, \langle\!\langle \rho_k \rangle\!\rangle$.

Observe that the preprocessing for [29] requires a circuit of size $O((m + \ell)n^3)$ to compute. Thus, the preprocessing material for [29] can be generated in communication $O(n^4|C|) + \mathsf{poly}(n)$ using [39].

M Concrete Example

In this section, we compare offline-online enhanced-$n/3$ protocol of ours against the honest-majority fully-secure protocol of [39]. The example below could be representative of US Treasury market auction (see `https://home.treasury.gov/services/treasury-auctions`).

For a circuit C, the total communication cost of the preprocessing protocol (as in Theorem 1.3 when instantiate with [39]) is $O(n^3 \cdot (|C| + L) \cdot \log|\mathbb{E}| + n^3\lambda^2 + n^6\lambda)$ bits plus $O(n^3 \cdot \lambda)$ bits in the assumed broadcast channel, where L is the number of (potential) clients that provide inputs.

Since the constant in big-O notation of [39] is 7.5, the above complexity can be concretely calculated to be $44 \cdot n^3 \cdot |C| \cdot \log|\mathbb{E}|$ and $15 \cdot n^3 \cdot L \cdot \log|\mathbb{E}|$. As an example, if $n = 32 = 2^5$, and $|C| = 2^{16}$, and $L = 2^{20}$, i.e. a circuit with 64K multiplication gates and number of clients of similar order, then, with $\log|\mathbb{E}| = 2^7$, we get the total communication to be around 78 terabits. On a 10 gigabits per second fiber-optic network, this large circuit needs preprocessing time of about 130 minutes (even without adjusting for parallel communication of the n parties). If we divide this by n (as all n parties can send their messages in parallel), this amounts to about 4 minutes.

For the same setting, where the base field $\log|\mathbb{F}| = 16$, we estimate the communication of the online phase of our protocol to be approximately $12 \cdot n \cdot |C| \cdot \log|\mathbb{F}|$, which is $12 \cdot 2^5 \cdot 2^{16} \cdot 16 = 3 \cdot 2^{27}$, which is 384 megabits. On the same 10 gigabits per second fiber-optic network, this will take 38.4 milliseconds. In contrast, the dominant term of [39] is the $O(n^6 \cdot \lambda)$ bits, which for the same setting can be estimated as $n^6 \cdot \lambda = 2^{30} \cdot 128 = 2^{37}$, which is 128 gigabits. This will take 12.8 seconds ignoring the additional required broadcast channel implementation. Note this is about 330 times slower than the online phase of our protocol.